Evolutionary Feature Manipulation in Unsupervised Learning

by

Andrew Lensen

A thesis submitted to the
Victoria University of Wellington
in fulfilment of the requirements for the degree of
Doctor of Philosophy
in Computer Science

Victoria University of Wellington
2019

I dedicate this thesis to my mum, Anne.
Even though you didn't get the chance to finish your thesis,
your love, training, and tenacity helped me to get through mine.

Abstract

Unsupervised learning is a fundamental category of machine learning that works on data for which no pre-existing labels are available. Unlike in supervised learning, which has such labels, methods that perform unsupervised learning must discover intrinsic patterns within data.

The size and complexity of data has increased substantially in recent years, which has necessitated the creation of new techniques for reducing the complexity and dimensionality of data in order to allow humans to understand the knowledge contained within data. This is particularly problematic in unsupervised learning, as the number of possible patterns in a dataset grows exponentially with regard to the number of dimensions. Feature manipulation techniques such as feature selection (FS) and feature construction (FC) are often used in these situations. FS automatically selects the most valuable features (attributes) in a dataset, whereas FC constructs new, more powerful and meaningful features that provide a lower-dimensional space.

Evolutionary computation (EC) approaches have become increasingly recognised for their potential to provide high-quality solutions to data mining problems in a reasonable amount of computational time. Unlike other popular techniques such as neural networks, EC methods have global search ability without needing gradient information, which makes them much more flexible and applicable to a wider range of problems. EC approaches have shown significant potential in feature manipulation tasks with methods such as Particle Swarm Optimisation (PSO) commonly used for FS, and Genetic Programming (GP) for FC.

The use of EC for feature manipulation has, until now, been predominantly restricted to supervised learning problems. This is a notable gap in the research: if unsupervised learning is even more sensitive to high-dimensionality, then why is EC-based feature manipulation not used for unsupervised learning problems?

This thesis provides the first comprehensive investigation into the use of evolutionary feature manipulation for unsupervised learning tasks. It clearly shows the ability of evolutionary feature manipulation to improve both the performance of algorithms and interpretability of solutions in unsupervised learning tasks. A variety of tasks are investigated, including the well-established task of clustering, as well as more recent unsupervised learning problems, such as benchmark dataset creation and manifold learning.

This thesis proposes a new PSO-based approach to performing simultaneous FS and clustering. A number of improvements to the state-of-the-art are made, including the introduction of a new medoid-based representation and an improved fitness function. A sophisticated three-stage algorithm, which takes advantage of heuristic techniques to determine the number of clusters and to fine-tune clustering performance is also developed. Empirical evaluation on a range of clustering problems demonstrates a decrease in the number of features used, while also improving the clustering performance.

This thesis also introduces two innovative approaches to performing wrapper-based FC in clustering tasks using GP. An initial approach where constructed features are directly provided to the k-means clustering algorithm demonstrates the clear strength of GP-based FC for improving clustering results. A more advanced method is proposed that utilises the functional nature of GP-based FC to evolve more specific, concise, and understandable similarity functions for use in clustering algorithms. These similarity functions provide clear improvements in performance and can be easily interpreted by machine learning practitioners.

This thesis demonstrates the ability of evolutionary feature manipulation to solve unsupervised learning tasks that traditional methods have struggled with. The synthesis of benchmark datasets has long been a technique used for evaluating machine learning techniques, but this research is the first to present an approach that automatically creates diverse and challenging redundant features for a given dataset. This thesis introduces a GP-based FC approach that creates difficult benchmark datasets for evaluating FS algorithms. It also makes the intriguing discovery that using a mutual information-based fitness function with GP has the potential to be used to improve supervised learning tasks even when the labels are not utilised.

Manifold learning is an approach to dimensionality reduction that aims to reduce dimensionality by discovering the inherent lower-dimensional structure of a dataset. While state-of-the-art manifold learning approaches show impressive performance in reducing data dimensionality, they do so at the cost of removing the ability for humans to understand the data in terms of the original features. By utilising a GP-based approach, this thesis proposes new methods that can perform interpretable manifold learning, which provides deep insight into patterns in the data.

These four contributions clearly support the hypothesis that evolutionary feature manipulation has untapped potential in unsupervised learning. This thesis demonstrates that EC-based feature manipulation can be successfully applied to a variety of unsupervised learning tasks with clear improvements in both performance and interpretability. A plethora of future research directions in this area are also discovered, which we hope will lead to further valuable findings in this area.

Acknowledgments

They say that it takes a village to raise a child — well, it takes a whole community to finish a PhD.

First and foremost, I must thank my supervisors, Prof. Mengjie Zhang, and A/Prof. Bing Xue. Without them, I would never have started postgraduate study, let alone come close to writing a thesis. Meng has an uncanny knack for putting issues into perspective, and for always keeping one eye on the bigger picture. Bing has a clear devotion to her students, being readily accessible at nearly any time of day to reassure or to give detailed and wise advice on any problem you may have. But most of all, Meng and Bing are two of the most compassionate and genuinely friendly people I have met — I am truly grateful for their supervision and friendship.

I wish to also thank all my friends and colleagues in the Evolutionary Computation Research Group (ECRG) and elsewhere in the university. Those in the ECRG are too numerous to mention, but I feel lucky to have such a supportive and friendly research group. I have also had the pleasure of fantastic officemates: Bach, Yanan, Qi, Truong, Victor, Yu, Yuxin, Boxiong, and Tony. You have all provided not only a much-needed distraction from the daily academic grind, but also have provided great feedback and stimulating discussion on my work. Many thanks to Damien, Diana, Pondy, Sarah, and all the others who have supported me during difficult times.

My appreciation for the support and love of my whānau cannot be understated. My Dad, Rob, has been the best father to me from my childhood through to today. I know that it must have been hard, but I would not be where I am today without your love and home-cooked meals. My siblings, Bridget and Daniel, have always looked after me and been amazing friends throughout my life.

To Liam: thank you for everything.

List of Publications

1. Harith Al-Sahaf, Ying Bi, Qi Chen, **Andrew Lensen**, Yi Mei, Yanan Sun, Binh Tran, Bing Xue, and Mengjie Zhang. "A survey on evolutionary machine learning". *Journal of the Royal Society of New Zealand*. 2019. Published online (May '19) [7].

2. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Genetic Programming for Evolving a Front of Interpretable Models for Data Visualisation". Submitted to IEEE Transactions on Cybernetics (August '19).

3. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Can Genetic Programming Do Manifold Learning Too?". In *Proceedings of the 22nd European Conference on Genetic Programming (EuroGP '19)*. Volume 11451 of Lecture Notes in Computer Science, pages 114–130. Springer, 2019. **Awarded best paper in EuroGP** [96].

4. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Genetic Programming for Evolving Similarity Functions for Clustering: Representations and Analysis". *Evolutionary Computation (Journal, MIT Press)*. 2019. Accepted, awaiting publication (October '19).

5. Damien O'Neill, **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Particle Swarm Optimisation for Feature Selection and Weighting in High-Dimensional Clustering". In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '18)*. pages 1–8. IEEE, 2018 [125].

6. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Automatically

Evolving Difficult Benchmark Feature Selection Datasets with Genetic Programming". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. pages 458–465. ACM, 2018 [94].

7. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Generating Redundant Features with Unsupervised Multi-tree Genetic Programming. In *Proceedings of the 21st European Conference on Genetic Programming (EuroGP '18)*. Volume 10781 of Lecture Notes in Computer Science, pages 84–100. Springer, 2018 [95].

8. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "New Representations in Genetic Programming for Feature Construction in *k*-means Clustering". In *Proceedings of the International Conference on Simulated Evolution and Learning (SEAL '17)*. Volume 10593 of Lecture Notes in Computer Science, pages 543–555. Springer, 2017 [92].

9. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Improving *k*-means Clustering with Genetic Programming for Feature Construction". In *Companion Material Proceedings of the Genetic and Evolutionary Computation Conference (GECCO Companion '17).* pages 237–238. ACM, 2017 [91].

10. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "GPGC: Genetic Programming for Automatic Clustering using a Flexible Non-hyper-spherical Graph-based Approach". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. pages 449–456. ACM, 2017 [90].

11. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Using Particle Swarm Optimisation and the Silhouette Metric to Estimate the Number of Clusters, Select Features, and Perform Clustering". In *Proceedings of the 20th European Conference on the Applications of Evolutionary Computation (EvoApplications '17)*. Part I, volume 10199 of Lecture Notes in Computer Science, pages 538–554. Springer, 2017 [93].

12. **Andrew Lensen**, Bing Xue, and Mengjie Zhang. "Particle Swarm Optimisation Representations for Simultaneous Clustering and Feature Selection". In *Proceedings of the Symposium Series on Computational Intelligence (SSCI '16)*. pages 1–8. IEEE, 2016 [89].

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Data mining [41] is a large and extensively studied area of Computer Science that attempts to discover *knowledge* in large datasets. Machine learning (ML) has been widely applied to data mining tasks such as classification, regression, and clustering. Supervised learning tasks, where data have expert-defined labels that can be exploited by data mining algorithms, have attracted the majority of attention in ML [75]. However, it has become clear that a significant proportion of the remaining challenges in ML cannot depend on learning from rich labelled data: many huge datasets cannot be feasibly labelled; other problems are so complex that even a human expert cannot solve them reliably [151]. Unsupervised learning, which seeks to discover intrinsic relationships or characteristics in unlabelled data, is seen as key to the future of artificial intelligence (AI) [152] by renowned ML experts such as Yann LeCun, who states: "We all know that unsupervised learning is the ultimate answer" and Jeff Dean, founder of Google Brain, who explains:

> "Supervised learning works so well when you have the right data set, but ultimately unsupervised learning is going to be a really important component in building really intelligent systems — if you look at how humans learn, it's almost entirely unsupervised."

One key challenge in data mining is producing useful results on datasets containing many attributes (*features*). For example, consider a (supervised) classification algorithm that uses all 100 features of a medical dataset to diagnose a patient. The first concern is how accurate this algorithm can really be, given the huge search space it was faced with optimising; secondly, and more critically, is the extent to which a doctor can trust the algorithm sufficiently given it is nearly impossible to understand such a complex model. These challenges are even more pronounced in unsupervised learning: as no labels are provided, the search space of possible solutions is strictly larger than in supervised learning; and the lack of labels makes validating and interpreting a complex unsupervised model even more futile.

Feature manipulation (FM) [98] transforms the input feature space of a dataset into a new (usually lower-dimensional) feature space containing features that are more concise in their information content, providing a smaller but more powerful set of features. FM methods are commonly categorised as *filter*, *wrapper*, or *embedded* methods. Filter methods directly evaluate a candidate transformed feature space using some feature quality measure; wrapper methods evaluate by utilising the results of feeding the new features into a ML algorithm; and embedded methods are those where the ML algorithm implicitly performs FM as part of its learning process.

FM has been widely applied to supervised learning tasks [160] with Evolutionary Computation (EC)-based methods showing particular success in recent years [120, 165, 179]. EC methods have been shown to be especially suitable, with their population-based approach allowing them to find sufficiently good solutions to a range of NP-hard problems in reasonable time [135]. However, there has been very little application of EC feature manipulation to unsupervised learning tasks — despite that these problems are strictly more difficult to solve than supervised tasks, and are of key importance to the future of machine learning. This can be seen through the number of results obtained when searching for

"supervised learning" or "unsupervised learning" in conjunction with "feature selection" or "feature construction" on common academic indexing services such as Google Scholar. As of September 2019, there are 64,900 papers that mention both "supervised learning" and "feature selection", but only 34,900 mentioning both "supervised learning" and "**un**supervised learning". When the 21,900 papers that mention both supervised and unsupervised learning are subtracted, there are only around 13,000 papers that focus on unsupervised learning. A similar result is found with "feature construction", with 1,690 papers focusing solely on FC in supervised learning compared to 460 in unsupervised learning. If we restrict the search criteria further to only include papers that use the term "evolutionary", there are fewer than 100 such papers mentioning EC, FC, and unsupervised learning.

1.2 Motivations

1.2.1 Challenges in Clustering

Clustering is the most-studied unsupervised task, which partitions a dataset into several groups (*clusters*), where each cluster contains similar or related instances of the dataset. For example, a group of people can be clustered by their age groups or according to their common interests. Clustering is widely used in many real-world applications [118].

A cluster partition that uses all features of a high-dimensional dataset to define its clusters is likely to be very difficult to analyse, as the instances contained in each cluster will be related to each other based on a very large number of attributes. A large proportion of clustering algorithms use distance-based techniques to evaluate how related two data points are; many distance functions, such as Euclidean distance, become increasingly less meaningful as dimensionality increases [1]. In addition, algorithms using many features tend to require significantly more computing power to perform clustering. In the worst case, using all features may mean the algorithm takes so long to finish that it is not a

viable choice for the user. Different features within a feature set often vary in their discernibility power — consider, for example, clustering historic weather records that contain a "day of week" attribute. While clustering the dataset by the day of week would produce a valid partition, the usefulness of the produced clusters is likely to be much lower than if a "rainfall (mm)" attribute was instead used.

A variety of different FM techniques have been proposed to combat these issues. Two common approaches used by algorithms are feature selection (FS), and feature construction (FC) [98]. While such FM techniques have been applied extensively to classification problems, they have seen relatively little use in clustering and other unsupervised learning algorithms.

Both clustering [40, 67] and FS/FC [10, 53] have been shown to be NP-hard problems. FS problems have a search space of 2^m possible feature subsets in a dataset with m features, and FC problems have an even larger search space, as every possible feature subset can be combined into a number of constructed feature(s) in many different ways. The most common FS techniques such as Sequential Forward Selection (SFS) [177] and Sequential Backward Selection (SBS) [108] fail to scale well when many features are present, as they tend to become stuck in local optima [138].

Clustering n instances into K groups can be performed S_n^K ways, which is a Stirling number of the second kind, defined as [50]:

$$S_n^K = \frac{1}{K!}\sum_{i=0}^{i=K}(-1)^{K-i}\binom{K}{i}i^n \tag{1.1}$$

The number of possibilities grows extremely quickly as n or K increases — for five instances being allocated to two clusters, there are 15 possibilities. However, for a slightly larger dataset, with $n = 20$ and $K = 2$, there are $524,287$ possible clusterings. One of the largest synthetic datasets commonly used in the literature has 2000 instances and 40

clusters — such a dataset can be clustered in 1.62×10^{3156} ways: clearly it is impossible to test all possible clusterings.

When K is unknown, the number of permutations is even bleaker at $\sum_{i=1}^{i=K_{max}} S_n^i$ where K_{max} is a pre-defined maximum number of clusters; in the extreme case $K_{max} = n$. As datasets continue to become increasingly longer (more instances) and wider (more features), fundamental clustering algorithms such as k-means fail to scale effectively [69]. Clearly, it is infeasible to find the best partition by trying every possible result on all but trivial problems. Hence, clustering methods have used a wide range of approximations and heuristics to find good clustering results in reasonable computational time.

EC [35] is an AI paradigm that contains stochastic population-based search techniques that supposedly draws inspiration from biological sources, such as evolution and animal swarming behaviour. Genetic Programming (GP) [82] and Particle Swarm Optimisation (PSO) [78] are two very successful EC methods, which been applied to a wide range of NP-hard problems successfully due to their ability to find sufficiently good solutions in reasonable time (i.e. hours to days).

The use of EC for clustering has been shown to produce superior results compared to existing deterministic algorithms [36, 64, 118]. However, even EC-based clustering struggles to perform well on datasets with a large number of features: more features introduce more ways to assign instances to clusters, ergo further increasing the search space size.

To address this, a few methods have been proposed for performing FS for clustering, and also for performing FC for clustering. However, these methods have a number of limitations, which are discussed in the following subsections. EC approaches for these tasks have not been studied nearly as comprehensively as in the supervised learning domain, where PSO for FS [179] and GP for FC [37] have seen substantial use in classification problems. Indeed, the use of FS with PSO for clustering has been recently suggested as a promising area for future research [36].

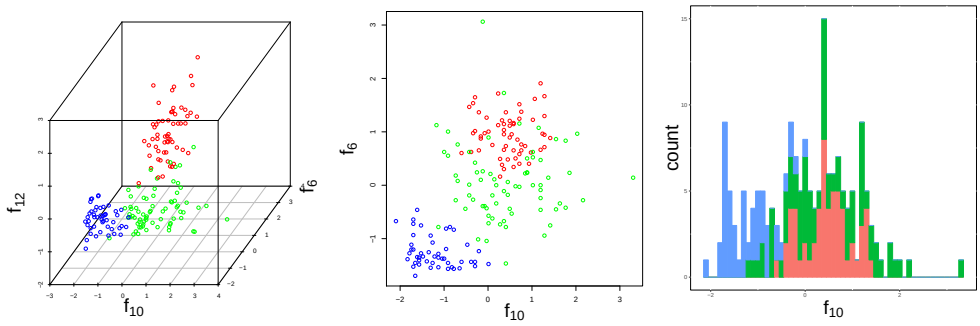
1.2.2 Limitations of Current EC Work for Feature Selection in Clustering

The majority of existing EC work combining FS with clustering tasks has used EC only to perform one of the two tasks of FS or clustering, with a non-EC method performing the other. Such an approach is inherently limited in that it does the two tasks separately; performing both simultaneously would allow the features chosen to be "tailored" to the clusters produced. Simultaneous FS and classification has been shown to be very effective [179], but simultaneous FS and clustering has been performed by only a few methods [71, 150], which have several limitations:

1. They either require K to be pre-defined [71] or use a variable-length encoding [150] where the length of the representation used is relative to K. Pre-defining K limits the usefulness of a method to cases where K is (approximately) known, and using a variable length encoding introduces difficulties in allowing individuals with different lengths to interact and exchange information.

2. They use fitness functions that linearly punish the number of features and clusters relative to the maximum number of features and clusters [71, 150]. Linearly punishing the number of clusters may give incorrect results as the EC process is likely to find it "easier" to decrease the number of features rather than increase the clustering performance. In a similar vein, applying a linear penalty based on K is likely to encourage bias towards small K, even on datasets that may actually have many clusters.

3. They have only been tested on small datasets with few features and clusters. FS is most useful on large, difficult datasets where reducing the number of features can significantly decrease computational resources required and the complexity of the solutions produced.



(a) 3 features: f_{10}, f_6, f_{12}. (b) 2 features: f_{10}, f_6. (c) 1 feature: f_{10}.

Figure 1.1: Wine dataset projected across varying numbers of features. Wine contains three classes, 13 features, and 178 instances.

There is also an inherent dependency between the number of features selected (m') and K. A larger m' will encourage a larger K and vice versa; the more information (i.e. features) available, the more easily the data can be divided into a larger number of smaller clusters instead of a few big clusters [8]. For example, consider the three plots shown in Figure 1.1, which show the Wine dataset projected using different numbers of features. The three colours represent the three classes of the Wine dataset. When three features are used in Figure 1.1a, it is easy to distinguish all three classes as distinct clusters. When one feature is removed in Figure 1.1b, the blue class still appears as a homogeneous cluster, however, the red and green classes are much closer and have enough overlap so that they may be considered as a single cluster. When only a single feature is considered in Figure 1.1c, all three classes overlap considerably and it is difficult to choose two thresholds that would split the three classes into three clusters well. As m' is minimised to encourage selecting fewer features, the evolutionary search will be biased towards picking smaller K, reducing performance on datasets that have large K.

A large number of fitness functions have been proposed for both clustering [118] and feature selection [179], but little research has been conducted into fitness functions that consider both clustering and feature subset quality. Both tasks require balancing several objectives: a good clustering partition will have clusters with high compactness, separability, and connectedness [56], whereas a good feature subset will

contain the smallest number of features possible that maximise performance. As the evolutionary process is directly guided by the fitness function used, it is important that a suitable fitness function is used in order to maximise the quality of the solutions produced.

1.2.3 Limitations of Current EC Work for Feature Construction in Clustering

Tree-based GP is an evolutionary algorithm that automatically models programs or functions using a tree-based structure, where the inputs of the program are the terminals of the tree, and the root is the output of the program. Tree-based GP has been used extensively for performing classification with feature construction built into the tree structure [37], but has only seen very limited use in clustering applications [118]. By using the feature set as terminals in a GP program design, feature selection and construction can automatically occur, as not all features are used in a tree, and function nodes operate on multiple features. Most existing GP methods for clustering perform feature selection and construction using an *embedded* approach, whereby feature manipulation occurs as a *consequence* of the program structure. This use of GP directly for cluster assignment may limit performance; GP is known to perform poorly when used directly as an embedded approach to many-class classification [37][1], which is an analogue to many-cluster clustering. Using a wrapper approach may produce better results by using an existing clustering algorithm to perform the clustering task. Such approaches have been successfully applied to the multi-class classification problem [120].

Much of the existing literature using GP for clustering encourages the formation of hyper-spherical clusters[2], either explicitly by using

[1] Due to the need to choose many thresholds that separate classes.

[2] Hyper-spherical clusters are those where instances lie in a hyper-spherical region around the cluster mean; in a 2D feature space, this produces circular clusters. Clusters need not be hyper-spherically shaped; a dataset may contain clusters of varying shape (e.g. elliptical, spiral, ring, etc. [69, 166]).

distance-based clustering techniques, or implicitly by using a fitness function that measures fitness in a hyper-spherical manner. Many datasets may not have hyper-spherical clusters at all, and GP's dynamic and flexible structure and representation should allow it to be used effectively for non-hyper-spherical clustering problems.

Beyond clustering, there are a range of other unsupervised problems that feature manipulation has the potential to address. The following subsections identifies and discusses two such problems: benchmark dataset creation and manifold learning.

1.2.4 Challenges in Creating Benchmark Datasets

The design of benchmark datasets is a growing research area in unsupervised learning; the lack of class labels makes it very difficult to evaluate the performance of unsupervised algorithms since there is no correct "ground truth". To tackle this, a range of benchmark clustering datasets have been produced [43], including using EC methods [56, 148] to produce complex, challenging clustering problems.

Another important data mining task that regularly uses benchmark datasets for comparison is feature selection. FS algorithms primarily work by identifying and removing *irrelevant* or *redundant* features from a feature set [160], either in a supervised or unsupervised manner. Irrelevant (or *noisy*) features are those that add little or no meaningful value to a dataset. In the worst case, an irrelevant feature may actually mislead the data mining process, when it contradicts the information given by other "correct" features. Removing such features reduces the search space of the data mining task, generally improving performance [99]. While removing irrelevant features is reasonably straightforward, redundant features (*r.fs*), which share a significant overlap of information content with other features, are much more difficult to identify, especially when they are redundant in a multivariate and/or interactive manner.

The most common method of quantitatively comparing the efficacy of different FS algorithms is to evaluate them on a range of popular datasets and compare the performance achieved and the number of features selected by each. Such an approach is quite a "coarse" analysis, as it gives little insight into which type of features each FS algorithm removes from a dataset. For example, naïve FS algorithms may readily remove clearly irrelevant features, while failing to identify r.fs completely. A more refined evaluation method is to directly look at how well each FS algorithm can remove each "type" of feature. However, identifying which features are non-linearly redundant in a dataset is an NP-hard problem (otherwise, the FS problem could be brute-forced!) [99]. Hence, an increasingly common technique is to purposefully add "new" irrelevant or redundant features to an existing dataset. While irrelevant features can be added with relative ease (choose a stochastic noise generator, and generate a number of noisy features), it is not obvious how to add features with complex redundancies.

The most naïve way of creating a r.f (Y) from a given *source* feature (X) is to multiply each feature value of X by some multiple α, such that the i^{th} value of Y is computed as $Y_i = \alpha X_i$ [52, 54, 181]. By varying α, one can easily generate any given number of r.fs based on X. A particularly straightforward method is to simply duplicate features (i.e. let $\alpha = 1$) — but these are trivial to remove. To make the redundancy weaker, one can introduce some bias (β) such as adding a constant value to each Y_i, e.g. $Y_i = \alpha X_i + \beta$. However, such approaches have a number of serious limitations.

The above types of r.fs have very simple redundancies that do not represent realistic interactions between features in real data mining problems. For example, in a dataset of people, two potential features may be an individual's age and income. It is generally true that the older a person, the more they earn, and so we may expect these features to be linearly redundant. However, a child is likely to have no income regardless of their exact age, and a pensioner is likely to have a similar

income to others aged over 65. While these two features are certainly partially redundant, the interaction is clearly more complex. In most datasets, the redundancy between two features tends to be even more complex. Removing r.fs that have linear redundancies is also quite a trivial FS problem, and so is not an adequate challenge for non-trivial FS algorithms. For example, a greedy algorithm that uses Pearson's correlation can easily find groups of linearly redundant features by measuring the correlation of each feature to those already selected.

There is hence an obvious need to have methods available to generate r.fs with (arbitrarily) complex interactions in order to benchmark FS methods more effectively. There has been very little work in the literature that has investigated how to automatically generate non-trivial r.fs. One common method used to automatically create functions to perform a particular task is GP. GP could be used to produce r.fs by taking a source feature as the program's input, and producing a r.f as the program's output. This could generate challenging FS datasets that have features with a range of redundancies, from simple univariate relationships to even more complex multivariate ones, i.e. many-to-many features.

1.2.5 Challenges in Manifold Learning

Manifold learning has risen to prominence in recent years due to significant improvement in autoencoders and the widespread use of the t-Distributed Stochastic Neighbour Embedding (t-SNE) visualisation algorithm [171]. Manifold learning is the main area in the non-linear dimensionality reduction literature, and consists of algorithms that seek to discover an embedded[3] (non-linear) manifold within a high-dimensional space so that the manifold can be represented in a much lower-dimensional space. Hence, they aim to perform dimensionality reduction while preserving as much of the *structure* of the

[3]Embedded in the manifold learning context means an underlying, existing, intrinsic structure in a dataset. Embedded in the feature manipulation context refers to a machine learning algorithm that performs FM as part of its learning process.

high-dimensional space as possible.

Popular manifold learning algorithms nearly exclusively work by producing an embedding in the low-dimensional space, which is represented by a set of points: these points are optimised freely according to some cost function, using a method such as gradient descent. For example, if the embedding had a dimensionality of two, then each instance in the dataset would have two feature values in the low-dimensional space, which are optimised starting from some point. While this approach has proved the most successful [111, 171], it has a critical limitation: there is no direct relationship between the high- and low-dimensional spaces. This is problematic in cases where it is important to apply the embedding to future examples (without re-running the whole algorithm), or even more restrictive when interpretability is important: since there is no clear relationship of the embedding to the original features, there is no meaningful way to interpret the embedding. As McInnes et al. note:

> "**For a number of uses [*sic*] cases the interpretability of the reduced dimension results is of critical importance. Similarly to most non-linear dimension reduction techniques (including t-SNE and Isomap), UMAP lacks the strong interpretability** of Principal Component Analysis (PCA) and related techniques such a [*sic*] NonNegative Matrix Factorization (NMF). In particular **the dimensions of the UMAP embedding space have no specific meaning**, unlike PCA where the dimensions are the directions of greatest variance in the source data. Furthermore, since UMAP is based on the distance between observations rather than the source features, it does not have an equivalent of factor loadings that linear techniques such as PCA, or Factor Analysis can provide. If strong interpretability is critical we therefore recommend **linear techniques** such as PCA and NMF." [111, p. 35]

Given that linear techniques are fundamentally limited in their ability to transform complex manifolds, and that interpretability is a key criterion

in many data mining applications, there is a clear need for new manifold learning techniques that are both powerful and interpretable. Manifold learning can be tackled as a feature construction problem, where the output embedding is a transformation of the original features. Given GP's ability to produce functional and interpretable mappings, it seems likely that GP techniques could be developed to tackle this challenge.

1.3 Goals

This overall goal of this thesis is to investigate the potential of using evolutionary unsupervised feature manipulation (FM) to improve the performance and interpretability of models in machine learning problems. This will be investigated through two main directions. First, the use of FM for improving the most common unsupervised data analysis task, clustering, will be explored. Second, the use of evolutionary FM to tackle unsupervised problems that evolutionary computation has not traditionally been applied to will be explored. These two directions will be investigated through two research objectives each.

The four specific objectives of this thesis are:

1. Investigate and develop a new PSO algorithm that performs clustering and *feature selection* simultaneously in the same particle, using new representations and refined fitness functions to improve clustering performance and select fewer features than existing methods.

   Existing work uses a restrictive centroid-based clustering encoding, and requires a variable-length encoding when K is unfixed [71, 150], which can often give invalid solutions and a difficult search space. Existing fitness functions also inherently bias decreasing dimensionality at the cost of reduced clustering performance due to the much smoother fitness space present in feature selection compared to clustering. The introduction of new clustering encodings and more refined fitness functions has the

potential to address these issues.

2. Design new methods for using GP to perform *feature construction* for clustering tasks. GP is widely known for its ability to improve performance in classification tasks by producing high-level constructed features from the original feature set, but the use of GP for clustering has been very limited. This also has the potential to improve the interpretability of cluster partitions through the use of fewer, more meaningful high-level features.

   GP-based feature construction wrapper methods have been very successfully applied to classification tasks [37], but have never been applied to clustering [118]. Wrapper methods are often quite powerful, as they can harness the efficiency/efficacy of an existing method by treating it as a "black box" into which constructed features can be fed. A GP wrapper method could be applied to an existing clustering method, such as k-means. Existing clustering methods require the selection of an appropriate (pre-defined) similarity function for use in the clustering process. However, these functions are nearly always inflexible, domain-agnostic, and require treating all features equally despite their varying importance. GP is well-known for its ability to evolve functions: GP-based feature construction has the potential to evolve custom similarity functions to improve the performance of clustering methods.

3. Develop an approach to creating challenging benchmark feature selection datasets. Evaluating the performance of FS algorithms comprehensively requires testing their ability to remove known redundant features. Common approaches such as duplicating or scaling existing features to introduce redundancy are not representative of challenging, real-world situations where multivariate complex redundancies exist.

   The creation of redundant features can be modelled as an unsupervised learning task, as redundancy is generally defined by

how feature distributions vary without considering any class information. Common measures, such as mutual information provide a proxy for measuring redundancy: by optimising to maximise mutual information, highly-redundant features can be designed. Such redundancies can be thought of as a redundancy function between the source (original) features and the newly created redundant features. Evolving this redundancy function is an application of feature construction, and so it is expected that GP is suited to this task. Furthermore, a multi-tree GP approach would provide a natural extension to creating multivariate (many-to-many) complex redundancies.

4. Propose the first GP-based approach to performing interpretable manifold learning. Manifold learning has become a very popular area of unsupervised machine learning research in recent years, with many high-performing approaches proposed.

   However, nearly all manifold learning algorithms have a glaring issue: they provide no understanding of how the manifold is formed from the original (high-dimensional) feature space. Interpretability is a key aspect of the data mining process, and arguably is most important on data that manifold learning is designed for: high-dimensional, complex data that humans cannot understand. Manifold learning can be seen as a type of unsupervised feature construction: the low-dimensional manifold can be created based on the high-dimensional structure present in the original features. Hence, GP could be used to evolve a functional mapping representing the manifold present in data. GP produces models that are inherently not black boxes — GP trees have the potential to be understood and interpreted, especially if parsimony pressure or other techniques are used. This is especially key in visualisation, where methods such as t-SNE give high-quality visualisations that cannot be related to the original feature set and so are limited in their usefulness.

1.4 Major Contributions

The major contributions of this thesis are as follows:

1. This thesis shows how a new medoid-based PSO approach can perform simultaneous feature selection and clustering in a single particle, utilising fewer features while achieving greater clustering performance compared to previous centroid-based PSO methods. This approach also allows the automatic discovery of the number of clusters (K) in a dataset while still using a more effective fixed-length representation. An advanced version of this algorithm that utilises a heuristic guide for K, and a pseudo-local search to fine-tune the best solution is also introduced; it shows further improvements across a range of benchmark datasets.

   Parts of this contribution have been published in:

   Andrew Lensen, Bing Xue, and Mengjie Zhang. "Particle Swarm Optimisation Representations for Simultaneous Clustering and Feature Selection". In *Proceedings of the Symposium Series on Computational Intelligence (SSCI '16)*. pages 1–8. IEEE, 2016.

   Andrew Lensen, Bing Xue, and Mengjie Zhang. "Using Particle Swarm Optimisation and the Silhouette Metric to Estimate the Number of Clusters, Select Features, and Perform Clustering". In *Proceedings of the 20th European Conference on the Applications of Evolutionary Computation (EvoApplications '17)*. Part I, volume 10199 of Lecture Notes in Computer Science, pages 538–554. Springer, 2017.

2. This thesis shows how the use of GP for feature construction can improve the performance and interpretability of existing clustering algorithms. A wrapper approach where GP is used to improve the performance of k-means is proposed, and is more accurate even when using only a few constructed features comprising a handful of original features. An alternative method, whereby GP is used to

evolve similarity functions for use by clustering methods, is also developed. This method automatically produces similarity functions that are specifically designed for the clustering algorithm and dataset being considered, which allows for more concise, meaningful, and powerful similarity functions compared to standard distance metrics such as Euclidean distance. Particular success is demonstrated when using a cohesive set of smaller similarity functions (i.e. multi-tree GP) to allow more tailored behaviour in different niches of the cluster space. The evolved similarity functions are shown to be significantly more interpretable than standard functions.

Parts of this contribution have been published in:

Andrew Lensen, Bing Xue, and Mengjie Zhang. "New Representations in Genetic Programming for Feature Construction in *k*-means Clustering". In *Proceedings of the International Conference on Simulated Evolution and Learning (SEAL '17)*. Volume 10593 of Lecture Notes in Computer Science, pages 543–555. Springer, 2017.

Andrew Lensen, Bing Xue, and Mengjie Zhang. "Improving *k*-means Clustering with Genetic Programming for Feature Construction". In *Companion Material Proceedings of the Genetic and Evolutionary Computation Conference (GECCO Companion '17)*. pages 237–238. ACM, 2017.

Andrew Lensen, Bing Xue, and Mengjie Zhang. "GPGC: Genetic Programming for Automatic Clustering using a Flexible Non-hyper-spherical Graph-based Approach". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. pages 449–456. ACM, 2017.

Andrew Lensen, Bing Xue, and Mengjie Zhang. "Genetic Programming for Evolving Similarity Functions for Clustering: Representations and Analysis". *Evolutionary Computation (Journal, MIT Press)*. 2019. Accepted with minor revisions (April '19).

3. This thesis shows how to generate redundant features using GP with a mutual information (MI)-based fitness function for benchmarking feature selection datasets. The proposed GPRFC method automatically creates difficult, redundant features that have the potential to be used for creating high-quality feature selection benchmark datasets. An extended multivariate approach (GPMVRFC) is also proposed; it is able to create more complex and realistic redundancy relationships between sets of multiple features. These methods provide a promising platform for further development of FS benchmark datasets.

   Parts of this contribution have been published in:

   Andrew Lensen, Bing Xue, and Mengjie Zhang. "Generating Redundant Features with Unsupervised Multi-tree Genetic Programming. In *Proceedings of the 21st European Conference on Genetic Programming (EuroGP '18)*. Volume 10781 of Lecture Notes in Computer Science, pages 84–100. Springer, 2018.

   Andrew Lensen, Bing Xue, and Mengjie Zhang. "Automatically Evolving Difficult Benchmark Feature Selection Datasets with Genetic Programming". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. pages 458–465. ACM, 2018.

4. This thesis shows how to use GP to perform manifold learning (GP-MaL). We have developed the first GP-MaL approach, which achieves competitive performance compared with existing manifold learning algorithms, while producing models that can be interpreted and re-used on unseen data. The value of GP-MaL is further reinforced through a new variation called GP-tSNE, which uses a multi-objective approach to produce visualisations that are both highly representative of the dataset while also being produced by interpretable GP trees. GP-tSNE is the first machine learning visualisation method that produces a front of visualisations that provide the user with a trade-off between visual clarity and model

interpretability. Deep insight can be gained when a set of solutions across a front are examined cohesively.

Parts of this contribution have been published in:

Andrew Lensen, Bing Xue, and Mengjie Zhang. "Can Genetic Programming Do Manifold Learning Too?". In *Proceedings of the 22nd European Conference on Genetic Programming (EuroGP '19)*. Volume 11451 of Lecture Notes in Computer Science, pages 114–130. Springer, 2019. **Awarded best paper in EuroGP.**

Andrew Lensen, Bing Xue, and Mengjie Zhang. "Evolving a Front of Representative and Interpretable Visualisations for Data Analysis". Submitted to IEEE Transactions on Evolutionary Computation (TEVC, March '19).

1.5 Organisation of Thesis

The contents of this thesis are organised as follows. Chapter 2 introduces a variety of background content and related work. Chapters 3 to 6 each address one of the research objectives and each produce one of the major contributions of this thesis. Chapter 7 concludes the thesis and highlights future research directions of interest.

Chapter 2 introduces fundamental background concepts for this thesis, including machine learning, unsupervised learning and clustering, feature manipulation, and evolutionary computation (PSO and GP). It also reviews existing related literature, highlighting limitations that this thesis aims to address.

Chapter 3 proposes a new PSO-based algorithm to perform clustering and feature selection simultaneously in a fixed-length PSO representation. An extended three-stage approach is then discussed, which offers further improvements while addressing limitations of existing PSO-based approaches. Experiments show clear improvements compared to existing approaches, with the proposed approach selecting fewer features while increasing clustering performance.

Chapter 4 develops two new approaches to using GP to perform FC for improving clustering performance. The first approach, which is the first approach to using wrapper-based GP for clustering, demonstrates how multi-tree GP can be used to significantly improve the performance of existing clustering algorithms. The second method proposed uses sophisticated techniques to automatically evolve custom similarity functions, which improve performance by tailoring similarity functions to the algorithm and dataset being used.

Chapter 5 establishes the first approach to automatically generating difficult FS benchmark datasets by evolving complex redundant features using multi-tree GP. An extension, which creates more realistic multivariate redundant features is also introduced. These methods are shown to create features that challenge existing FS algorithms and showcase the clear potential for using GP-based FC to tackle challenging and previously unexplored unsupervised learning.

Chapter 6 introduces the first usage of GP for performing interpretable manifold learning by using GP-based FC techniques. The benefits of this approach compared to existing manifold learning algorithms is clearly demonstrated: the proposed approach achieves competitive performance using a model that can be interpreted in a straightforward manner. This is further demonstrated through the application of this approach to visualisation tasks, where the transparency of the models evolved by the proposed approach are shown to increase the amount of knowledge provided by the generated visualisations.

Chapter 7 summarises the main contributions of this thesis, highlighting the key findings of each chapter. A number of wider conclusions are also provided for the future use of EC-based feature manipulation in unsupervised learning. This thesis then concludes by discussing a range of specific potential areas of future research.

Chapter 2

Literature Review

This chapter begins by introducing fundamental concepts of machine learning relevant to this thesis, including unsupervised learning and clustering. Core concepts of feature manipulation are then provided, followed by an overview of evolutionary computation, and mutual information. Related work on using evolutionary computation for feature manipulation and unsupervised learning is discussed, with a focus on clustering due to the lack of existing use of evolutionary computation for manifold learning and feature creation.

2.1 Machine Learning

Machine Learning (ML) is a central paradigm in Artificial Intelligence (AI), which has progressed dramatically during the 21^{st} century and is now commonly regarded as the best approach for tackling a plethora of real-world tasks [75]. Unlike traditional algorithms — that are programmed to perform pre-defined actions — ML algorithms *learn* to solve a task/problem by examining data and automatically building and refining a *model* to achieve the best possible result [9]. There is a huge amount of diversity in ML algorithms, but they are commonly grouped by the way in which the algorithm receives *feedback* to iteratively refine its model. The most typical categorisation of techniques is whether they perform supervised, unsupervised, or reinforcement learning [145]:

- In supervised learning tasks, each instance (datum) is provided to the ML algorithm with a pre-defined desired output (label), which the algorithm should learn to reproduce [115]. Supervised ML algorithms must learn a function/model that correctly maps inputs (features of the instance) to the desired outputs. Supervised learning is historically the most commonly studied ML paradigm, with tasks such as classification (categorical outputs) and regression (continuous outputs) being extensively researched.

- Unsupervised learning tasks, in contrast, provide **no labels** for the ML algorithm to use to facilitate learning. In lieu of labels, unsupervised ML techniques attempt to find underlying patterns or characteristics of the data [60]. By far the most well-known approach is clustering, but in recent years, research interest in unsupervised feature learning and non-linear dimensionality reduction has surged in the research community [14].

- In reinforcement learning tasks, desired outputs are not directly provided, however, *rewards* or *punishments* are provided to the learner depending on the actions it takes [159]. In this way, the ML algorithm implicitly learns based on the decisions it chooses to make. Reinforcement learning has become increasingly well-known in recent years from DeepMind's reinforcement learning approaches giving similar or better-than-human performance in Atari games [114], and Go and Chess [151].

This thesis focuses solely on unsupervised learning tasks. More specifically, it considers the well-established task of clustering, as well as the two more recent and lesser-studied tasks of redundant feature creation and interpretable manifold learning. A background to these tasks and related concepts is provided in the next three sections.

2.2 Clustering

Clustering is an important part of exploratory data mining that aims to group similar items (instances) of a dataset together into a number (K) of *natural* groups (clusters) [46, 69]. Unlike in supervised learning, where labels are used to evaluate performance, the quality of a clustering solution (partition) is measured by using one or more of a wide range of measures [116, 118]. Many different types of clustering algorithms, which are effective on a range of different datasets with different properties and clustering objectives, have been proposed [3].

A huge variety of approaches have been proposed for performing clustering [3, 178], which can be generally categorised into hard, soft (fuzzy), or hierarchical clustering methods. In hard and soft clustering, each instance belongs to exactly one or to at least one cluster respectively. In contrast, hierarchical clustering methods build a hierarchy of clusters, where a parent cluster contains the union of its child clusters. The majority of work has focused on hard clustering, as partitions where each instance is in exactly one cluster tend to be easier to interpret and analyse. A number of distinct models have been proposed for performing hard clustering: prototype-based models (including the most famous clustering method k-means [68], and its successor k-means++ [12]), density-based models (e.g. DBSCAN [38] and OPTICS [11]), and graph-based models [146] (e.g. the Highly Connected Subgraph (HCS) algorithm [59]). Statistical approaches, such as distribution-based models (e.g. Expectation–maximisation (EM) clustering [21]) and kernel-based models [48] have also been proposed. Each of these common categories of clustering algorithms are explained in more detail below.

2.2.1 Common Clustering Algorithms

Prototype-based clustering: Prototype-based clustering algorithms produce a number of *prototypes*, each of which corresponds to a distinct

cluster centre, and then assigns each instance to its nearest prototype using a distance function, such as Euclidean distance. While these models are the most popular, they are inherently limited by their use of prototypes to define clusters: typically they can only be used to produce hyper-spherical clusters due to optimising by minimising a distance function to all instances in the cluster. However, clusters need not be hyper-spherically shaped [69, 166]; a variety of valid cluster shapes are shown in Figure 2.1.

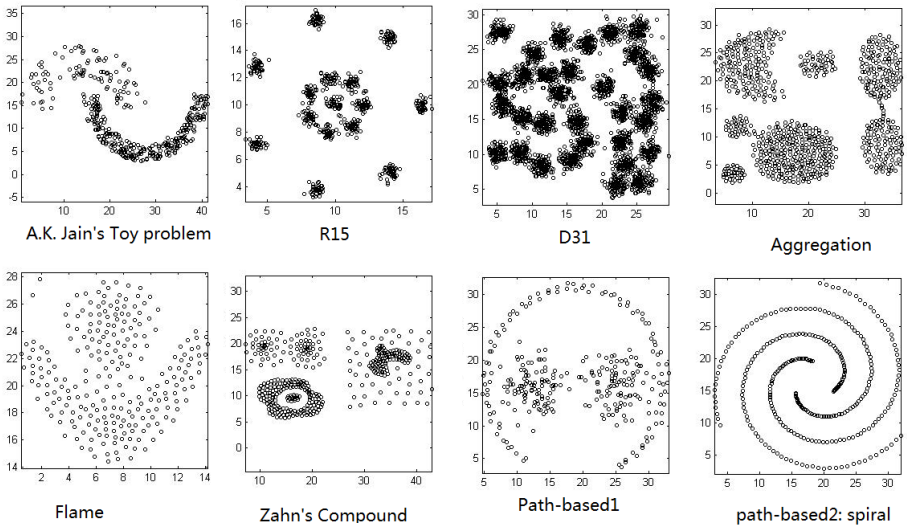


Figure 2.1: Hand-crafted datasets exhibiting a range of geometries and densities [43].

***k*-means** [68] is the canonical example of a prototype-based clustering algorithm. The original k-means algorithm generates K initial cluster centroids (prototypes) randomly in the feature space. Each instance in the dataset is then assigned to the nearest cluster centre using a distance measure such as Euclidean distance. The centres of each cluster are then recomputed by finding the mean of all instances in the cluster. Each instance is then again assigned to its nearest cluster and cluster centres are recomputed. This process continues until a number of iterations is reached or until the clusters stabilise. The performance of k-means is highly dependant on the quality of the initial cluster centroids. While

k-means is efficient and can have good performance when its assumptions are met, it has a number of limitations: K must be pre-defined; the clusters produced are very sensitive to the initial centres chosen; when K is high, performance tends to decrease; and the clusters produced will tend to be compact but may not be well-separated or well-connected.

k-means++ [12] is an improved version of k-means, which performs more intelligent selection of the initial (seed) values for clusters by probabilistically encouraging the selection of seeds that are far apart. k-medoids is an alternative approach, which enforces the constraint that cluster centres must be instances in the dataset (*medoids*) [127], which is useful in ML in situations where a single instance from each cluster should be selected (e.g. instance selection).

Density-based Clustering: Density-based clustering algorithms (e.g. DBSCAN [38]) define clusters as being regions of high density and label instances in sparse regions as outliers [3]. DBSCAN has been largely succeeded by OPTICS [11]. OPTICS improves upon DBSCAN by detecting clusters in data of varying density more accurately, i.e. by ordering instances by their similarity. Unlike DBSCAN, OPTICS does not explicitly produce a partition; instead a parameter, ξ, is used — the value of ξ represents the relative decrease in density that represents a cluster boundary, e.g. $\xi = 0.1$ corresponds to a 10% drop in density.

Density-based clustering algorithms often suffer from their need to choose parameter values carefully: picking poor values may lead to many small dense clusters, which should be merged, or to only a few big clusters that are overly general and should be split further. They are also known to scale poorly in high dimensions: instances become increasingly equidistant in higher dimensions [15] (the "curse of dimensionality"), which leads to increasingly uniform density across the feature space and a lack of dense regions representing clusters.

Hierarchical Clustering: Hierarchical clustering algorithms build a hierarchy of clusters, merging or splitting clusters at different levels of the hierarchy. These generally fall into one of two types: agglomerative methods, where each instance starts in its own cluster and clusters are repetitively joined as you move up the hierarchy; and divisive methods, which use a contrary approach where all instances start in a single cluster and are split into increasingly smaller clusters [142]. The output of hierarchical clustering is a *dendogram*, which can be cut at any given point in order to give a clustering partition. The best manner in which to choose a cut is often unclear or is domain-specific and is an active area of ongoing research. Common techniques include requiring the user to choose a pre-defined K, using clustering quality or information theoretic-measures [112], or using visualisation techniques [147].

Agglomerative clustering has seen greater research efforts due to its significantly lower computational cost: it is much cheaper to decide which two small clusters to merge than how to split one big cluster into an exponential number of two sub-clusters. Single-linkage and complete-linkage clustering are canonical agglomerative clustering methods that pick the two nearest clusters to combine based, respectively, on the closest or furthest instances between the two clusters. A third, less common approach is average-linkage clustering, where the distance between the means of each cluster is used [153].

Graph-based Clustering: Graph-based clustering algorithms [174] represent clusters as distinct graphs, where there is a path between every pair of instances in a cluster graph. Graph-based approaches are seen as an intuitive way of modelling clustering problems, as instances (nodes) should generally be in the same cluster as their nearby neighbours; in other words, close instances will have an edge between them. This representation means that graph-based measures are not restricted to clusters with hyper-spherical or convex shapes. The popular HCS algorithm [59] uses a similarity graph that connects instances sharing a similarity value (e.g. distance) above a certain threshold. It then

iteratively splits graphs that are not *highly connected* by finding the minimum cut, until all graphs are highly connected. Choosing a good threshold value in HCS can be difficult when there is no prior knowledge of the data.

Subspace Clustering: Another sub-paradigm of clustering research is *subspace clustering* [101, 116], where each cluster is located in a subspace of the data, i.e. it uses only a subset of the features. In this regard, each cluster is able to correspond to a specific set of features that are used to characterise that cluster, which has the potential to produce better-fitted and more interpretable clusters. Subspace clustering methods can be seen as implicitly performing dimensionality reduction.

2.2.2 Measuring Clustering Performance

As clustering is an unsupervised learning task, solutions cannot be evaluated using a test set as in supervised learning (with the exception of synthetic data). As a result, a number of measures have been proposed to evaluate the goodness of a given cluster partition. These measures may consider a range of criteria, as follows:

Compactness: how tightly-packed a cluster is. Clusters should be as compact as possible, so as to ensure that only the most related instances have been grouped together. A cluster's compactness can be measured in a number of ways, such as each instance's distance to its nearest/furthest cluster neighbour, to all cluster neighbours, or to the cluster mean.

Separability: how well neighbouring clusters are separated in the feature space. A partition with clusters that are far apart is a good partitioning as instances have been clearly separated into clusters. As with compactness, a range of approaches can be used, e.g. the distance from each cluster to the nearest/furthest other cluster, or to the dataset mean.

Connectedness: instances that are close together by distance should generally be allocated to the same cluster as they have similar characteristics. Maximising connectedness encourages the formation of clusters that better represent the dataset. Unlike the previous two criteria, connectedness is generally measured per-instance rather than per-cluster. The most common approach used is to find the mean distance from each instance to its n-nearest neighbours.

Number of clusters: when the expected number of clusters is known in advance, it is obvious that the best solutions will contain the correct number of clusters. When the expected number of clusters is between some minimum and maximum or completely unknown, it can be difficult to evaluate how many clusters should be formed. A common heuristic is to limit the number of clusters to between 2 (the smallest valid partition) and $\sqrt{n}$, where n is the number of instances [126]. This heuristic can fail on small datasets that have many distinct instances, and so require many clusters. This heuristic also gives a very wide range of possible K, which introduces a very large search space for large n.

A good partition will be maximally connected, minimally sparse, and maximally separated, and have K clusters, where K is some optimal number of clusters.

The most common clustering measures consider either or both of the intra- (compactness) and inter-cluster (separability) distances when evaluating a partition.

2.2.3 Clustering Measures

A number of common clustering measures are defined and explained below. Across all measures, n and m represent the number of instances and features respectively, and K represents the number of clusters in a given partition. C_i and Z_i represent the i^{th} cluster and the mean of the i^{th} cluster respectively. The dataset mean is given by Z^*. The distance $d(a, b)$ between two instances can be calculated using any distance function. The

most prevalent distance function is Euclidean ("straight-line") distance:

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_m - b_m)^2} \tag{2.1}$$

where a_i and b_i give the i^{th} feature value of instances a and b.

Internal Measures

The internal measures listed below are ones that evaluate cluster quality based purely on the properties of the partition produced by a clustering algorithm [46]. Each measure is labelled with either a $\uparrow$ or a $\downarrow$ to indicate it should be maximised or minimised respectively.

1. Sum intra-cluster distance:

$$\text{Intra}_{\text{Sum}} \downarrow = \sum_{i=1}^{K} \sum_{a \in C_i} d(a, Z_i) \tag{2.2}$$

   Minimising the sum of the intra-cluster distances will give compact clusters. If partitions are evaluated using only this measure, then the best partition will occur when every cluster contains a single instance (i.e. $n = K$), as each cluster will have no intra-cluster variation. Hence, this measure is most suitable when K is pre-defined or as a component in a more complicated measure.

2. Root Mean Squared Error:

$$\text{RMSE} \downarrow = \sqrt{\frac{1}{K} \sum_{i=1}^{K} CSE_i^2} \tag{2.3}$$

   where

$$\text{CSE} = \sqrt{\frac{1}{|C_i|} \sum_{a \in C_i} d(a, Z_i)^2} \tag{2.4}$$

   RMSE is very similar to the sum intra-cluster distance measure, but better punishes poor-quality clusters with instances far away from

the cluster mean.

3. Sum *inter*-cluster distance:

$$\text{Inter}_{\text{Sum}} \uparrow = \sum_{i=1}^{K} \sum_{j=1}^{K} d(Z_i, Z_j) \tag{2.5}$$

Maximising the sum of the inter-cluster distances will give well-separated clusters. A number of variations to the above equation are used, including summing the distance from each cluster mean to the dataset mean, i.e. $d(Z_i, Z^*)$ or from each cluster mean to the closest neighbouring cluster mean, i.e. the minimum of $d(Z_i, Z_j)$ across all $j \neq i$. A third variation instead considers the distance between the two closest points of each cluster pair:

$$\text{Inter}_{\text{minDistSum}} \uparrow = \sum_{i=1}^{K} \sum_{j=1}^{K} \min_{a \in C_i, b \in C_j} dist(a, b) \tag{2.6}$$

The best variation of this measure will often depend on the dataset or the learning algorithm used.

4. Davis-Bouldin index [28]:

$$\text{Davies-Bouldin} \downarrow = \frac{1}{K} \max_{1 \leq i < j \leq K} \frac{S_{C_i} + S_{C_j}}{dist(Z_i, Z_j)} \tag{2.7}$$

where

$$S_{C_i} = \frac{1}{|C_i|} \sum_{a \in C_i} d(a, Z_i) \tag{2.8}$$

The Davies-Bouldin index measures the ratio of within-cluster scatter (i.e. intra-cluster distance) to inter-cluster separability. The two clusters that have the highest ratio give the output of the Davies-Bouldin index. This measure derives a partition's quality from the two worst clusters in that partition, meaning it may give an overly *pessimistic* view of the clustering partition when two clusters are much worse than the rest of the clusters. A lower value

of this index corresponds to a better distance.

5. Dunn index [32]:

$$\text{Dunn Index} \uparrow = \frac{\min_{1 \leq i < j \leq K} dist(Z_i, Z_j)}{\max_{1 \leq i \leq K} \max_{a,b \in C_i} dist(a, b)} \tag{2.9}$$

The numerator in Equation (2.9) finds the minimum distance between any two clusters. The denominator finds the maximum distance between any two instances that are in the same cluster. Hence, by maximising the numerator and minimising the denominator, a higher value of the Dunn index will correspond to a better quality partitioning. This measure is similar to the Davies-Bouldin index in that it considers the inter-cluster distance of the two closest clusters.

6. Silhouette Criterion [143]:

The silhouette criterion measures how well a given instance is matched to its cluster. It is defined as follows:

$$Silhouette(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{2.10}$$

where $a(i)$ is the average distance between instance i and all other instances in its cluster; $b(i)$ is the *minimum* average distance between instance i and the instances in each other cluster. A silhouette value of 1 indicates an instance is perfectly clustered; a value of -1 indicates it should be in a neighbouring cluster; a value of 0 indicates it is on the border of two clusters. The average silhouette computed across all instances in a partition gives a measure of how good the partition is, and implicitly balances both the intra- and inter-cluster measures.

7. Scatter trace criterion [45]:

$$\text{Scatter} \uparrow = trace(S_W^{-1} S_B) \tag{2.11}$$

This criterion uses scatter matrices[1] to give a measure that considers both within-cluster scatter (S_W, compactness) and between-cluster variation from the dataset mean (S_B, separability).

External Measures

While clustering is an unsupervised learning task, it can be helpful to test the performance of a clustering method on a dataset where the clusters have previously been labelled — a partition can be evaluated based on how accurately it reproduces the labelled partition. This comparison is complicated as the number of clusters produced need not be the same as the labelled partition. Furthermore, there is no general method for mapping clusters between the labelled partition and generated partition, and so clusters cannot be directly compared in terms of their memberships. Measures that perform such a comparison are called external measures, as information (i.e. labellings) external to the partition produced is used to evaluate their performance. A variety of external measures have been proposed:

1. Purity measures: A good cluster will contain instances from only a single class. The purity across the clusters in a partition can hence give an indication of how good that partition is. Sheng et al. [150] computed the classification error in the following way:

   (a) For each cluster, find the majority class label of the instances in that cluster.

   (b) Count the number of misclassified instances, where an instance is misclassified if it is not the majority class.

   (c) Find the total error rate as the fraction of misclassified instances across the whole partition.

[1] A scatter matrix is an estimation of a covariance matrix, used where a full covariance matrix is overly expensive or infeasible to create.

2. Another common external evaluation measure is the Rand index (RI) [140]. This index measures the similarity between two clusterings; here it is used to measure the similarity between the known classification and the cluster produced. For every pair of instances in the dataset, the Rand index checks two things: if the pair are in the same cluster, and if the pair are in the same class. From this, four values are computed:

   (a) a is the number of pairs that are in the **same** cluster and in the **same** class.

   (b) b is the number of pairs that are in different clusters and in different classes.

   (c) c is the number of pairs that are in the **same** cluster but in different classes.

   (d) d is the number of pairs that are in different clusters but in the **same** class.

   The Rand index is then computed as follows:

$$Rand \uparrow = \frac{a+b}{a+b+c+d} \tag{2.12}$$

   This index essentially measures the fraction of agreements between the clustering solution and the known classifications. Pairs of instances should only be in the same cluster if they are in the same class, and vice versa.

   The Adjusted Rand Index (ARI) [124] is a corrected-for-chance version of the Rand Index, which corrects the RI by accounting for the expected similarity of all pair-wise comparisons between clusters in a random model [173]. The ARI is most easily understood through a formulation using a contingency table:

   Given a cluster partition C produced by an algorithm and a gold standard cluster partition G, the ARI is calculated by first generating a contingency table where each entry n_{ij} denotes the

number of instances in common between C_i and G_j, where C_i is the i-th cluster in C, and G_j is the j-th cluster in G. In addition, the sum of each row and column is computed, denoted as a_i and b_j respectively. As before, n is the total number of instances. The ARI is then calculated according to Equation (2.13), which finds the frequency of occurrence of agreements between the two cluster partitions, while adjusting for the chance grouping of instances.

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{2.13}$$

3. Cura [27] used an error rate criterion to measure the similarity between a partition and the known correct classification. This criterion uses a similar approach to the Rand index (and is, in fact, equivalent) but instead produces a percentage error rate across the instances in the dataset:

$$ER \downarrow = \Big[\frac{2}{n(n-1)} \times \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} disagree(I_a, I_b)\Big] \times 100\% \tag{2.14}$$

where $disagree(I_a, I_b)$ is 0 if Instances I_a and I_b are either both in the same classes and same clusters or in different classes and clusters, and 1 otherwise.

4. In this thesis, a variation of the F-measure used in supervised learning is also used for measuring clustering performance due to its familiarity. We adapt the F-measure used in classification tasks, by considering each pair of instances in turn (as it is not possible to directly decide if an instance is in the "right" cluster) and choose which of the following cases apply:

   (a) Same class label, belong to the same cluster: true positive (TP).

   (b) Same class label, belong to the **different** clusters: false negative (FN).

(c) **Different** class labels, belong to **different** clusters: true negative (TN).

(d) **Different** class labels, belong to the same cluster: false positive (FP).

The F-measure is then calculated in the normal way using the total number of TPs, FPs, and FNs, as follows:

$$\text{F-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.15}$$

$$\text{precision} = \frac{TPs}{TPs + FPs} \tag{2.16}$$

$$\text{recall} = \frac{TPs}{TPs + FNs} \tag{2.17}$$

The F-measure uses a similar approach to the RI, but weights precision and recall using a harmonic mean rather than an arithmetic one. It is also particularly suited to accurately representing results on imbalanced datasets, i.e. where clusters vary greatly in their size.

2.2.4 Estimating K

Clustering methods can also be broadly categorised based on whether they require K to be pre-defined, or if they are able to automatically determine an acceptable K in the learning process. While the latter case is more flexible and hence is generally more useful, it may happen that a domain expert is available and is able to determine an optimum K, in which case the former methods are likely to perform better as they search a much smaller search space.

A wide range of statistical techniques for estimating the number of clusters (K_{est}) in a given dataset have been proposed [24]. While studies have compared the efficacy of many techniques [24], there is no consensus on the best technique for the general case. An overview of two

of the most popular methods, the Gap Statistic [164], and the use of the Silhouette Criterion [76] are discussed below.

The Gap Statistic

The Gap Statistic method estimates K_{est} by performing clustering on the dataset across a range of K values and then comparing the clustering partitions to those produced when clustering is performed on reference datasets generated using a uniform distribution [164]. The Gap value, $Gap(K)$, for a given K is computed by comparing the sum of the intra-cluster distances (i.e. sum error (SE)) of the dataset partition to that of each of the reference dataset partitions. K_{est} is then chosen as follows:

$$K_{est} = \min\{K \geq 2 | Gap(K) \leq Gap(K+1) - sd_{K+1}\} \tag{2.18}$$

where sd_K is the standard deviation of the Gap values across the reference datasets. In other words, the gap statistic attempts to find the smallest K which produces a clustering partition with the lowest intra-cluster variation relative to what would be expected on a randomly sampled reference dataset.

A range of clustering methods can be used within the Gap Statistic method; the performance of the clustering method will directly affect the K_{est} produced. k-means is commonly used, but Partition around Medoids (PAM) is generally regarded as having consistently better performance, albeit at a significantly larger computational cost. While the Gap Statistic produces reasonably accurate K_{est} on most datasets, it requires a large amount of computational time on larger datasets due to the need to generate and cluster many reference datasets. Pham et al. [133] proposed a method that uses similar principles to the Gap Statistic but instead employs a heuristic estimate of the intra-cluster variance of the reference data. This estimate greatly reduces the computation time required to give K_{est}, even when PAM is used for clustering. Empirical testing indicates a similar accuracy compared to the Gap Statistic.

The Silhouette Criterion for Estimating K

The silhouette criterion can be used to give K_{est} by performing clustering for each potential K and then choosing the K for which the average silhouette is highest [76]. This can be computationally expensive due to the need to compute the pair-wise distance between all instances in a dataset. However, this computation only needs to be performed once at the start of the algorithm.

2.3 Feature Manipulation

Feature manipulation (FM) is the act of purposefully altering the feature set of a dataset in order to improve the outcomes of a machine learning algorithm. FM is most often used to improve learning efficiency and performance and/or improve the interpretability of the models/solutions produced by decreasing model complexity. In statistical domains, FM is often instead referred to as *dimensionality reduction* (DR). It can be argued that FM, unlike DR, can actually increase the number of dimensions in a dataset; if new features are added alongside the original features, there will be a higher dimensionality. However, this approach is not used in this thesis, and so FM and DR are treated as interchangeable terms here. The two most common categories of feature manipulation are feature selection (FS) and construction (FC) [99]. FS attempts to select an optimal smaller subset of features in order to improve performance and decrease complexity, whereas FC improves performance by creating new, more powerful meta-features that combine multiple features in some way.

FM methods are also often categorised by how they evaluate the quality of the manipulated features. The most common categorisation is into wrapper, filter, hybrid, and embedded methods [8]. Wrapper methods use a learning algorithm to evaluate the quality of a manipulated feature set and choose the one that gives the highest performance on the learning algorithm. Filter methods take a different approach where the quality of

a manipulated feature set is measured more explicitly using a measure such as information gain or entropy [53]. Filter methods tend to give inferior results to wrapper methods, but are usually quicker in terms of computational time required [100]. Filter methods can also be performed purely as a pre-processing step to the main data mining task, making them independent to the learning algorithm being used. Hybrid approaches combine both filter and wrapper methods to give better performance than filter methods while being quicker to run than wrapper methods. Embedded approaches perform FM directly as part of the learning algorithm being used, and so can be designed efficiently while being tailored to the algorithm being used; however, they tend to be more algorithm-specific.

While FS approaches are sufficient for datasets that have a large number of redundant and/or redundant features, their ability to reduce dimensionality on other data is limited. For example, if we want to reduce the dimensionality to two or three features, using FS alone is likely to poorly retain the structure of the dataset. In such a scenario, FC approaches have the potential to retain more structure/information of the original feature set; however FC has a strictly larger search space and the constructed features are likely to be harder to understand than a subset of original features. Both of FS and FC are discussed in further detail in the following subsections, including a survey of relevant existing work.

2.3.1 Feature Selection

Feature selection, the task of selecting a subset of features in order to decrease complexity and increase performance, is perhaps the most popular feature manipulation task. Many FS algorithms have been proposed, the simplest of which is likely filter-based feature ranking [102], where features are ranked according to their quality according to a measure such as class label correlation (in supervised learning) or inherent information content based on Mutual Information

(MI). The top n features are then chosen as the best n features to use for the given dataset. While such an approach is efficient, simple, and algorithm-agnostic, it does not consider the *interactions* between features. In the trivial case where two features in a dataset are identical and of high-quality, feature ranking would naïvely select both despite the second being redundant. In other words, "the n best features are not the best n features" [70].

The well-known minimal-redundancy-maximal-relevance criterion (mRMR) [131] refines this concept by minimising the redundancy between selected features, as well as maximising the relevance with respect to the class label/optimisation criteria. However, given there are 2^n possibilities for selecting n features, using this more sophisticated approach introduces a new problem: how can this NP-hard search space be effectively optimised? The simplest approach is to use a sequential forward or backward selection (SFS, SBS) technique [138, 177]. SFS begins with no features selected, and repetitively greedily selects the next unselected feature that maximises the feature selection criterion. SBS takes an opposite approach, starting with all features selected, and repetitively removing the feature that improves the criterion the most [2]. In the case of mRMR, this corresponds to greedily selecting a feature subset that gives the most information possible while selecting features that are least-redundant. As SFS/SBS are greedy approaches, such an approach addresses the computation complexity issues — but as a greedy search is used there is no guarantee that the best (or even a good) subset can be found. Variants of SFS/SBS such as Sequential Floating Forward/Backward Selection (SFFS/SFBS) [138] have been proposed, but these have their own limitations (e.g. how much back-tracking to perform in a floating search). Other more advanced methods such as using sparse models [17, 22] have been proposed. These sequential searches can also be used as wrapper-based methods, by using the performance of a ML algorithm as the criterion for measuring the quality

[2] Or decreases the FS criterion the *least*, depending on how the problem is formulated.

of a feature subset.

One of the most common embedded FS approaches is the use of regularisation to penalise the creation of models with many coefficients; this encourages models to be formed with zero-valued coefficients, which correspond to unselected features. This approach is commonly used in regression, such as in the least absolute shrinkage and selection operator (LASSO) method [163], but has also been used in Support Vector Machines (SVMs) [186] and Decision Trees (DTs) [30] for classification. Other ML methods also implicitly perform FS as part of their learning process, such as in DT learning where often not all features will be used in an attempt to reduce over-fitting [79].

Feature selection applications have been extensively studied on a range of problems including classification in supervised learning [160], and in a range of unsupervised learning problems [33] such as clustering [8]. In recent years, increasing focus has been applied to the use of Evolutionary Computation (EC) for FS in these domains due to their strength in tacking NP-hard tasks: this will be discussed in detail in Section 2.6.1.

2.3.2 Feature Construction

Feature construction techniques take a different approach to FS, whereby they attempt to construct new, higher-level features that are transformations and combinations of multiple features in the dataset [154]. The new constructed features are trained to have improved performance, while also reducing the number of features used by the machine learning algorithm. The use of FC can improve interpretability by automatically combining useful features in an understandable manner.

One of the most well-known FC methods is Principle Component Analysis (PCA) [74]. PCA produces *components* (constructed features) that are linear combinations of the original features, such that each successive component has the largest variance possible while being

orthogonal to the preceding components. Variance is a fundamental measurement of the amount of *information* in a feature, and so PCA is optimal for performing linear dimensionality reduction under this framework. However, linear combinations are not sufficient when data has a complex underlying structure; linear methods tend to focus on maintaining global structure while struggling to maintain local neighbourhood structure in the constructed feature space [171].

SVMs [26] are a supervised learning algorithm that construct a (set) of hyperplanes, which maximally separate distinct classes in the data. These hyperplanes are formed as combinations of the existing feature vectors, and so are considered by some to be a form of FC. Some work has been done to interpret the meaning of SVMs through examining the weights used in a model to identify relevant features [139, 156]. DTs can also be considered to be an embedded FC approach, given that they determine an instance's class using complex meta-rules based on the instance's feature values.

As in FS, there has recently been a surge in research using EC techniques for FC tasks. In particular, Genetic Programming has seen significant success due to its model structure being well-suited for FC [154]. EC-based FC techniques will be discussed in further detail in Section 2.6.2. Another increasingly popular form of FC is methods that perform nonlinear dimensionality reduction, which is recently more commonly called manifold learning [88].

2.3.3 Information Theory: Mutual Information

Mutual Information (MI) [72] is an important concept in the field of Information Theory. MI is used as a way to measure the amount of information shared by two variables (or features). In this way, it is a measure of the mutual dependence of two variables, and is one way to measure how redundant one feature is with respect to another — the higher the MI, the more redundant the features are said to be. MI is formalised as follows:

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{2.19}$$

where the entropy of a feature X, $H(X)$, is defined as:

$$H(X) = -\sum_{x \in X} p(x) \times \log_2 p(x) \tag{2.20}$$

and the joint entropy of two features, X, Y, is:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \times \log_2 p(x,y) \tag{2.21}$$

Equation (2.19) can be expanded as follows:

$$MI(X,Y) = -\sum_{x \in X, y \in Y} p(x,y) \times \log_2 \frac{p(x,y)}{p(x)(y)} \tag{2.22}$$

The above definition of MI assumes that the two features have discrete values; in the case of continuous features (such as in this work), the below definition applies:

$$MI(X,Y) = \int_X \int_Y p(x,y) \times \log_2 \frac{p(x,y)}{p(x)(y)} dx\, dy \tag{2.23}$$

Calculating the MI of two continuous features requires knowing the full marginal and joint probability density functions (*pdf*) of the two features. In practice, this is infeasible, as the feature values for a given feature can be thought of as only a sample of the underlying *pdf* [83]. As such, a number of MI estimators have been proposed for estimating the MI of two continuous features. One venerable method proposed by Kraskov et al. [83] uses a nearest-neighbour estimation approach, which compares the similarity of neighbours for each instance across the two dimensions X and Y as a proxy to gauge the strength of the relationship between X and Y. This approach, implemented in the Java Information Dynamics Toolkit (JIDT) [103], is employed throughout this thesis.

It is often more common to compare sets of related features — for example, two sets of selected features. In such a case where there is a many-to-many (i.e. multivariate) relationship, an extension of Equation (2.23) is used, where X and Y represent the set of source and target features respectively, e.g. **X** and **Y**. A similar equation is used, except that the marginal and joint probabilities represent the set of features, i.e. $p(\mathbf{X}) = p(X_1, X_2, ...X_d)$ for d source features. As with univariate MI, continuous multivariate MI is approximated using Kraskov's approach as part of the JIDT tooklit.

2.4 Manifold Learning

Manifold learning (MaL) algorithms are based on the assumption that the majority of real-world datasets have an intrinsic redundancy in how they represent information they contain through their features [14]. A manifold is the inherent underlying structure that contains the information held within that dataset, and often this manifold can be represented using a smaller number of features than that of the original feature set due to the local "flatness" present in the geometry of the dataset structure [14]. Thus, MaL algorithms attempt to learn/extract this manifold into a lower-dimensional space. PCA, for example, can be seen as a linear MaL algorithm; of course, most real-world manifolds are strongly non-linear [14]. While MaL can be considered to be FC, it is discussed separately here as it is generally considered to be its own field of research.

A common categorisation of MaL algorithms is whether they perform an *explicit* or *implicit* mapping from the high- to low-dimensional space [88]. An explicit mapping method (also known as an "embedding" method) directly produces a low-dimensional representation for each instance, which means that generalisation to new instances is not possible, and interpretability with respect to the original high-dimensional representation is often infeasible. In contrast, an implicit mapping method (also known as simply a "mapping" method) creates a

parametrised function between the high- and low-dimensional spaces. Such methods have the benefits of being re-usable on future data, and also the potential to be understood by humans. However, finding such a mapping is inherently more difficult than directly optimising a low-dimensional representation, as the function space cannot be optimised through traditional methods such as (stochastic) gradient descent. In this thesis, the terms embedding method and mapping method are used to highlight the key difference between directly finding an embedding, and evolving a functional mapping. State-of-the-art MaL methods are nearly exclusively of the embedding type.

Multidimensional Scaling (MDS) [84] was one of the first approaches to MaL, and attempts to maintain between-instance distances as well as possible from the high- to the low- dimensional space. Metric MDS often uses a loss function called *stress*, which is then minimised using a majorizing function from convex analysis. Another well-known, more recent method is Locally-Linear Embedding (LLE) [144], which describes each instance as a linear combination of its neighbours[3], and then seeks to maintain this combination in the low-dimensional space using eigenvector-based optimisation. MDS performs a non-parametric transformation of the original feature space, and so is not interpretable with respect to the original features; LLE is also difficult to interpret given it is based on preserving neighbourhoods.

Self-organising maps (SOMs) [80, 81] are a type of two-layer artificial neural network, which can be regarded as a MaL algorithm, as well as a clustering method. SOMs map the high-dimensional input space to a (generally) two-dimensional *region* of output nodes, which can be interpreted as a two-dimensional visualisation. Each output node of the SOM is fully connected to the input layer, and so has m weight values for m features in a dataset. These weight values can be used to calculate the distance between the input and output spaces for a given instance. These weight values are optimised such that close output nodes in the SOM

[3]Here, neighbours refer to the closest instances to a point by (Euclidean) distance.

will have similar distances to the same instances. In this way, different parts of the SOM output layer will correspond to different inherent groupings of the dataset. SOMs are commonly used due to their relatively simple approach, and ability to create visually-understandable outputs. As a standard SOM is fully-connected, it scales poorly as the input dimensionality increases, and is directly affected by the curse of dimensionality. While the outputs of SOMs can be interpreted, the mapping itself from high- to low-dimensional space is opaque due to the large number of weights ($\#outputs \times m$) required for datasets with significant complexity.

t-Distributed Stochastic Neighbour Embedding (t-SNE) [171] is considered by many to be the state-of-the-art method for performing visualisation (i.e. 2D/3D MaL); it models the original feature space as a joint probability distribution in terms of how close an instances' neighbours are and then attempts to produce the same joint distribution in the low-dimensional space by using Kullback–Leibler divergence to measure the similarity of the two distributions. However, t-SNE was developed purely for visualisation (2/3D dimensionality reduction) and so it is not specifically designed as a general MaL algorithm [171]. It is also similar to MDS in that it produces an embedding with no mapping back to the original features. Finally, autoencoders are often regarded to do a type of MaL [14], but again they tend to be quite opaque in the meaning of their learnt representation, while requiring significantly more computational resources than the classical MaL methods.

2.4.1 Manifold Learning for Visualisation

Visualisation is a fundamental task in data mining that aims to represent data in a human-understandable graphical manner [41]. The simplest commonly-used machine-learning visualisation methods are FS and PCA. By using FS to select only two (or perhaps three) features, one can visualise a dataset by plotting the feature values of each instance along the x- and y-axes. PCA can be used for visualisation by plotting the first

two principle components: this gives the optimal visualisation that can be produced by using only linear transformations. While using FS clearly allows for ease understanding of the produced visualisation, it is extremely limited in its efficacy when the feature space is non-trivial due to the use of only two/three raw features. PCA (and other linear mapping techniques) also have the potential to be interpreted, but are inherently limited in the quality of their output dimensions on large datasets by their use of linear weightings only.

Creating optimal non-linear transformations is an NP-hard problem, with many machine learning methods proposed as a result. The earliest methods include techniques such as Isomap [161] and Local Linear Embedding (LLE) [144]. SOM have also been extensively used for visualisation [49, 132], but are limited in understandability by their fully-connected structure.

t-SNE [171] is generally regarded as the mainstream state-of-the-art visualisation method. t-SNE is an improvement to the previously proposed SNE method [62], which introduced a more nuanced probabilistic mapping from the high- to low-dimensional spaces based on the similarity of neighbours in the high-dimensional space. t-SNE improved upon SNE by introducing a cost function that could be better optimised by gradient-based optimisers; and by using a heavy-tailed t-distribution in place of a Gaussian to address the tendency of points in the low-dimensional space to be pushed together at the cost of separation between points, a characteristic known as the *crowding problem*. A range of improvements to t-SNE have since been proposed, including a more efficient tree-based nearest-neighbour search [170], and a parametric version [169].

Parametric t-SNE [169] is a variation of t-SNE that allows for out-of-sample re-use of the learnt t-SNE representations on future samples. Parametric t-SNE constructs a mapping from the high- to low-dimensional space using restricted boltzmann machines to construct a pre-trained feed-forward neural network model. The neural network

architectures used over 10,000 neurons on the biggest dataset, which heavily restricts the potential for interpreting the mapping that the neural network represents.

Autoencoders [63] are another neural-network based approach, which attempt to compress the representation of the data into the narrowest middle hidden layer as possible such that the original data can be re-created from the concise representation. To do so, autoencoders use a number of layers of varying sizes to encode and decode the data. This provides a mapping to and from the learnt representation, but again it is unrealistic to interpret given the number of nodes and fully-connected topology. Clearly, the solutions found by these methods are opaque to humans and provide little "intuitive" understanding of the data. Even the most recent advances such as Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [111] acknowledge significant weaknesses in regards to interpretability.

2.5 Evolutionary Computation

Evolutionary Computation (EC) [35] is a research area of Artificial Intelligence (AI) that includes stochastic population-based search techniques that draw inspiration from nature. EC methods are generally classified into one of three paradigms: evolutionary algorithms (EAs) [44], which are motivated by evolutionary principles; swarm intelligence (SI) methods [77], which are motivated by animal swarming behaviour; and other methods that are based on some other metaphor [13]. EC methods iteratively improve an initial set (*population*) of solutions (*individuals*) by measuring the quality (*fitness*) of each; using the best individuals to create new, hopefully superior individuals, which replace the population in the next iteration (*generation*); and then repeating this process until stopped. Genetic Programming (GP) [82] and Particle Swarm Optimisation (PSO) [78] are very successful EA and SI methods respectively, which been applied to a wide range of NP-hard problems successfully, due to their ability to find sufficiently good

solutions in a reasonable amount of time. This thesis uses GP and PSO extensively; they are each discussed in detail in the following subsections.

2.5.1 Genetic Programming

GP [82] is an EA technique where individuals are modelled in the form of computer-style "programs". The most common representation uses a tree-based structure, where the root of the tree is the output of the program, the leaves of the tree are inputs, and the internal nodes (i.e. non-leaves) are *functions*. Each function in a tree performs a certain task, by taking a number of inputs and producing an output based on these inputs. A common function is the "$-$" operator, which takes two floating-point values as input, and then outputs the result of subtracting the second input from the first. The leaves of the trees are called *terminals*, as they take no inputs, but produce some output. Common examples of terminals include features from the dataset, or constant values.

As GP is an EA, it uses mechanisms inspired by biological evolution to iteratively refine (*evolve*) the individuals, in its population. The GP process begins by randomly generating enough individuals to fill the population. It then runs for a number of iterations (*generations*); in each generation, the following steps are performed:

1. Measure the fitness of all individuals in the population using the fitness function.
2. Choose a number of the best individuals according to their fitness, using a *selection* method (e.g. tournament selection).
3. Apply genetic operators (mutation and crossover) to the chosen individuals to create new individuals, in an attempt to improve them.
4. Replace the worst individuals in the current population with the newly created individuals.

5. (*Optional*): ensure the best solutions from the current generation are maintained in the new population (*elitism*).

6. Repeat Steps 1 to 5 until the total number of generations is reached, or another termination criteria (such as optimal fitness) is fulfilled.

Genetic Operators

Each of the genetic operators are applied to a given selected individual probabilistically according to the parameters set by the user. For example, 20% mutation would mean that 20% of the time, a selected solution has mutation applied to it; the other 80% of the time, another operator is used. The *mutation* operator generally selects a random sub-tree of the program, and then replaces it entirely with a new randomly-generated sub-tree, which is generated in the same manner as how the initial population was generated. In doing so, there is a chance that a tree will be improved. The *crossover* operator randomly selects two compatible sub-trees in two distinct individuals, and swaps them, to give two new individuals. Crossover is the mechanism through which individuals in the population can exchange "good" partial solutions ("building blocks"), and is one of the main mechanisms that allows GP to outperform random search techniques. Reproduction simply takes a solution and inserts it unchanged into the new population; it differs from elitism in that it is performed probabilistically on all selected solutions rather than simply transferring the n best solutions.

Multi-tree GP

A variant of GP is multi-tree GP [117], where each GP individual contains t trees instead of a single tree. This is particularly useful for ML problems that benefit from heterogeneous behaviour/niching, or which have several distinct sub-problems to be optimised. There are a variety of approaches to performing crossover and mutation in multi-tree GP, but a common approach is to pick a tree within an individual at random, and then apply the genetic operator to that tree as in single-tree GP. Multi-tree

GP will be used extensively in this work with FC to represent multiple constructed features in a single candidate solution.

2.5.2 Particle Swarm Optimisation

PSO is an SI technique inspired by bird flocking behaviour [78]. In PSO, the swarm (population) consists of a number of particles (individuals), which explore the search space based on their personal experience as well as the swarm's social knowledge of good areas of the search space. Each particle encodes a single solution to the problem being optimised as a position vector, which represents the particle's position in the search space. This position vector takes the form $[x_1, x_2, ..., x_D]$, where D is the number of dimensions in the search space. The velocity of a particle encodes its movement through the search space and is represented using a vector of the form $[v_1, v_2, ..., v_D]$. During each iteration of the PSO search process, a particle's velocity is updated based on its previous best position ($pbest$) and the swarm's overall best known position ($gbest$). The particle then updates its position based on its current velocity, giving an updated position and hence a new candidate solution. The position and velocity updates are formally defined as follows:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2.24}$$

$$v_{id}^{t+1} = w \times v_{id}^{t} + c_1 r_1 \times (p_{id} - x_{id}^{t}) + c_2 r_2 \times (p_{gd} - x_{id}^{t}) \tag{2.25}$$

where t is the t^{th} iteration of the PSO process and $d \in D$ is the d^{th} dimension of the search space. The inertia weight, w, is used to balance the exploration and exploitation behaviour of the particles. c_1 and c_2 are acceleration constants that balance the contributions of the $pbest$ and $gbest$ positions respectively. r_1 and r_2 are randomly generated values in $U[0, 1]$ that introduce variance to the search process. p_{id} and p_{gd} are used to represent the values of $pbest$ and $gbest$ in the d^{th} dimension. The updated velocity, v_{id}^{t+1}, is restricted to the range $[-v_{max}, v_{max}]$ for some maximum velocity, v_{max}, in order to limit oscillation.

2.5.3 Evolutionary Multi-Objective Optimisation

Multi-objective optimisation (MO) is a technique used when a problem intrinsically has two (or more) *conflicting* objectives between which a trade-off must be made by any solution to the problem. The quality of a solution in this context is often relative to the objective function values of other solutions. For example, given two candidate solutions y, z to a problem that has two minimisation objectives, then the following test can be used to determine if y is strictly better than (or *dominates*) z:

$$\forall i : f_i(y) \leq f_i(z) \text{ and } \exists j : f_j(y) < f_j(z) \tag{2.26}$$

where $i, j \in 1, 2, 3, ..., k$ for k objectives. A solution that is not dominated by any other solution is called a *non-dominated* solution. For a given problem, the Pareto set contains all the Pareto-optimal solutions: those that are not dominated by any other possible solution to the problem. This set is mapped to the objective space by the *Pareto front*, which is the subset of solutions that represent the best trade-offs between the different objectives. MO algorithms attempt to find the set of non-dominated solutions which give the best approximation of the Pareto front.

EC is one of the most successful and widely used approaches to MO as its population-based structure naturally allows for multiple non-dominated solutions to be found in a single run. Of the Evolutionary Multi-objective Optimisation (EMO) algorithms, the non-dominated sorting genetic algorithm (II) (NSGA-II) [29] is perhaps the most popular for two-objective problems and is still regarded as a very strong and simple method despite being proposed in 2002.

NSGA-II works by constructing a series of dominance rankings, whereby an individual's ranking is based on the number of other solutions that dominate it. Non-dominated solutions have a ranking of 0, with other solutions having higher (worse) rankings respectively. These rankings are then used as "fitness" in the evolutionary process, with selection

methods such as tournament selection using these rankings to select solutions for breeding. The concept of *crowding distance* is used to encourage diversity of solutions, by discouraging the selections of solutions with close neighbours. The final output of the EC process is a set of unique, non-dominated solutions (i.e. the best approximation of the Pareto front).

A plethora of other EMO methods have been proposed [185]. This thesis uses EMO solely with GP. Multi-objective genetic programming [16, 23] most commonly utilises NSGA-II. While other EMO methods may give better results, this thesis is not focused on exploring EMO in depth, and so we use NSGA-II as a known reliable EMO method.

2.6 EC for Feature Manipulation

Research into the use of EC techniques for performing feature manipulation has become much more popular during the last decade, due to the ability of EC techniques to efficiently search in a large feature set space. Despite this, the use of EC for feature manipulation in unsupervised learning tasks has thus far been relatively unexplored.

2.6.1 EC for Feature Selection

EC techniques have been used widely for feature selection [47, 179], with PSO and Genetic Algorithms (GAs) being used for filter, wrapper, and hybrid approaches. These two algorithms are both well-suited for FS tasks as they use a vector encoding, where each feature can be represented by a single value, which determines if that feature is selected. Genetic Programming (GP) has also been used for performing embedded feature selection [117]. EC has seen some use for FS in clustering tasks, which will be surveyed extensively in Section 2.7.2.

2.6.2 EC for Feature Construction

Tree-based GP has emerged as the predominant FC technique due to its dynamic model structure allowing features to be combined in a hierarchical manner using a variety of powerful functions [37, 120]. Most work uses a representation where a single GP tree produces a single constructed feature, as the output of the tree. The input to the tree is generally the set of features, and an optional random value input. This representation has been extended so that multiple features may be constructed in a single GP individual, commonly using a multi-tree representation [117]. Other representations have also been proposed [37], including using multiple sub-trees as a set of constructed features [4, 165], using specially-tailored node designs [184], cooperative co-evolutionary GP [97], and even by performing multiple GP runs (each producing a single constructed feature) [120].

GP-based FC has been used in classification [165], image analysis [6], and other supervised domains [4, 51, 58], but has seen nearly no use in unsupervised learning tasks. The limited work using GP for clustering tasks mostly performs feature construction as a "side-effect" of using features as terminals, rather than explicitly aiming to utilise GP for feature construction to improve performance. These approaches will be discussed in detail in Section 2.7.3.

2.6.3 EC for Manifold Learning and Visualisation

EC has seen very recent use in manifold learning through evolving autoencoders for image classification tasks using Genetic Algorithms [158], GP [141], and PSO [65, 157]. Historically, autoencoders have had to be manually designed or required significant domain knowledge to get good results, and so automatic evolution of the structure of autoencoders with high performance is a clear improvement. However, these methods are still a somewhat indirect use of EC for representation/manifold learning, as they do not allow an EC method to

directly learn the underlying manifold of the data. Success is also fundamentally limited by the need for neural network architectures that are differentiable. GP has also been used for visualisation in a supervised learning context using a multi-objective fitness function to optimise both classification performance and clustering-based class separability measures [23]. The use of GP for visualisation of solutions for production scheduling problems has also been recently investigated [123].

Multi-objective GP approaches have been proposed to improve the visualisation quality of feature construction in classification problems. The Multi-objective Genetic Programming Projection Pursuit (MOG3P) algorithm [66] aimed to produce solutions with a trade-off between three objectives: classifiability, visual interpretability, and semantic interpretability, by using the multi-objective Strength Pareto Evolutionary Algorithm 2 (SPEA-2) [187]. The three objectives used do not appear to be strictly conflicting: on a naïve classification dataset, it would be possible to achieve perfect classification performance with a simple hyperplane (or line), which could be represented by a simple GP tree that produces a visualisation with two very distinct classes. Even on more complex datasets, there is an inherent relationship between the visual interpretability of a dataset (i.e. how well classes are separated), and the classification performance; given well-separated classes, one would expect correct classification to result. A later approach focused on classifiability and visual interpretability only, but used a number of different measures for each of these objectives [23].

Tree-based GP is often recognised for its ability to directly model functions by taking inputs as leaves and producing an output at the root [135] and is often used for dimensionality reduction in the form of FC [120]. Significant progress has been made on producing interpretable GP trees, which are simple enough to be understood by a human expert by using techniques such as parsimony pressure [135] and multi-objective optimisation [18], with multi-objective approaches showing particularly good results due to their ability to produce a range

of solutions with different levels of complexity [23, 176]. Despite being clearly suitable for dimensionality reduction and evolving interpretable trees, GP has never been applied as a MaL technique to produce interpretable models that create understandable visualisations.

2.7 EC for Clustering

EC techniques have also been applied to clustering successfully [46, 104, 118, 134, 149] with many GA and PSO techniques used to automatically evolve clusters. Most EC techniques have focused on performing hard partitional clustering (where every instance is assigned to exactly one cluster) [64, 118] with the majority of techniques using a prototype-based approach. Clustering performance is generally measured in terms of connectedness (how well neighbouring instances are assigned to the same cluster), compactness (how densely packed a cluster is) or separability (how well clusters are separated from each other) [64]. EC-based clustering methods using vector (PSO/GA) encodings and tree-based (GP) encodings are surveyed in the following subsections, as these are the most related to this topic of research. In addition, work using EC for FS with statistical feature grouping techniques are discussed, as this is an application of EC-based clustering to a FS task.

2.7.1 PSO and GA for Clustering

Both PSO and GA have seen extensive use for clustering [36, 64]. Both algorithms tend to use related approaches, due to their similar representations. Most methods can be broadly defined by the use of one of the three following encoding schemes [64].

Centroid Encoding

The centroid encoding scheme directly encodes the co-ordinates of cluster centroids into the particle or chromosome representation. For

example, a single centroid using five features may have the following representation: $[0.20, 0.60, 1.15, 0.70, 2.30]$. This encoding gives a length of mK for m features and K clusters (where K must be pre-defined). Each instance is generally assigned to the cluster with the closest centroid (i.e. 1-nearest neighbour). A downside of this encoding is that its length will grow linearly with m, giving a very large search space on big feature sets. Hence, the application of feature selection to this representation should be able to allow it to be better applied to datasets with many features. This is by far the most prevalent encoding scheme used.

Medoid Encoding

A medoid-based system uses an encoding of length n for n instances, where each instance is assigned a single value [64]. A high value (e.g. 1 in a binary encoding) for an instance corresponds to that instance being a medoid, i.e. a cluster centre. Hence, the number of clusters is defined by the number of medoids selected. All instances that are not medoids are assigned to the cluster with the closest medoid, usually by Euclidean distance. A benefit of this scheme is that the number of clusters does not have to be pre-set by the user, a useful characteristic in clustering algorithms. However, the search space will grow quickly as the number of instances increases, which can be an issue in large datasets. The medoid representation is restricted to choosing cluster prototypes that are instances in the dataset. As a result, it may not be able to perform exploitation (i.e. local searching) as effectively as the centroid representation, which can produce centroids located at any point in the search space. This is less likely to be an issue on datasets where there is a large number of instances as there will be a higher level of granularity in terms of the medoids available.

Label Encoding

The third common design uses a labelling system, where each instance is assigned a label representing the cluster it belongs to. A similar

representation to the medoid encoding is used, where each instance of the n instances has a single value in an encoding of length n. The value for an instance directly gives its cluster label. This design is more flexible than the medoid or centroid schemes, as it does not use a distance measure to define a cluster membership. In other words, a cluster is not encouraged to be hyper-spherical, allowing clusters to form shapes that are suitable for the dataset. Such an encoding can produce multiple redundant solutions, e.g. the solutions $[1, 1, 2, 2, 3, 3]$ and $[2, 2, 1, 1, 3, 3]$ produce the same cluster partition. This increases the search space unnecessarily unless a renumbering procedure is used. Furthermore, it can be difficult to design a system that will give a smooth search space when using a labelling encoding; if integers are used to label instances, then a PSO system will have to round position values. The multi-objective clustering with automatic k-determination algorithm (MOCK) [56] uses a GA with a label-based graph approach to perform multi-objective clustering. Another GA method has also been proposed, which takes inspiration from spectral clustering and uses either a label-based or medoid-based encoding to cluster the similarity graph [113].

2.7.2 PSO or GA with Feature Selection for Clustering

While a range of EC techniques have been proposed for either clustering or FS, only a few of these techniques perform simultaneous clustering and FS. All of these techniques use a centroid encoding [71, 150] to perform clustering, and a binary encoding for performing FS. An example of such a representation is shown in Figure 2.2 for m features and K clusters. The first m dimensions correspond to the m features in the dataset. The value for a given feature determines if it has been selected. The remaining Km dimensions correspond to the co-ordinates of the K cluster prototypes.

The most influential work in the EC domain for simultaneous clustering and feature selection is the Niching Memetic Algorithm for simultaneous Clustering and Feature Selection (NMA_CFS) [150], which uses a GA to

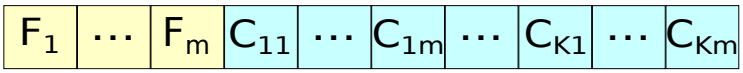


Figure 2.2: Centroid representation for simultaneous clustering and feature selection.

simultaneously perform FS, find K, and cluster the dataset. The authors use niching and local search techniques to improve the performance and consistency of the method. The niching approach requires a number of parameters to be set in order to optimise performance. NMA_CFS uses a variable-length encoding where the solution size is relative to the number of clusters produced. While this method produced impressive results compared to existing algorithms, the datasets used for testing contained a small number of clusters (a maximum of 6 and 7 in the synthetic and real-world datasets respectively) and a small number of features (m) with a maximum of 20 and 30 on synthetic and real-world datasets respectively. In practice, datasets may have many more clusters and features; it is not obvious how well NMA_CFS would scale as K and m increase.

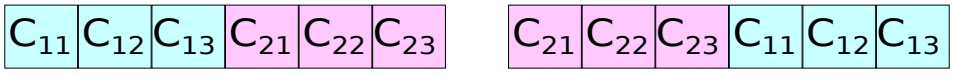
The NMA_CFS algorithm used a fitness function based on the scatter trace criterion. The J1 criterion (see below) applies a penalty weighting based on the number of features selected by the clustering algorithm, which encourages the use of fewer features. The clustering algorithm is hence encouraged to find a trade-off between two criteria: clustering accuracy and the number of features used. The J1 criterion may limit the performance because of this simple weighting technique; using more advanced compositions of these two criteria or a multi-objective approach may improve results. Sheng et al. [150] also introduced the J2 criterion, which applies an additional penalty weighting to the J1 criterion based on the number of clusters formed by a given solution. This additional weighting seems somewhat arbitrary — encouraging a small number of clusters is not correct on all datasets, and hence may lead to a smaller number of clusters being formed than the optimal number.

In the below equations, $m^{'}$ represents the number of selected features, and K_{max} is defined to be $\sqrt{n}$, as per the original definitions [150].

$$J1 \uparrow = trace(S_W^{-1} S_B) \times \frac{m - m^{'}}{m - 1} \tag{2.27}$$

$$J2 \uparrow = trace(S_W^{-1} S_B) \times \frac{m - m^{'}}{m - 1} \times \frac{K_{max} - K}{K - 1} \tag{2.28}$$

The use of a variable-length encoding may also reduce training effectiveness due to the inability of candidate solutions to fully exchange information when they vary in length. A centroid encoding also introduces an implicit ordering to the clusters in the solution when clusters do not have any ordering in reality. For example, consider the two solutions shown in Figure 2.3 — both solutions will produce the same cluster partition, but as their two centroids are in different orders, the training process will incorrectly attempt to move each solution towards the other. In a GA approach, for example, this could mean that crossover produces a solution that contains conflicting or even duplicate centroids.



(a) Solution A (b) Solution B

Figure 2.3: Two centroid solutions with different centroid orderings that produce the same partition.

Javani et al. [71] proposed a PSO-based method that uses a similar centroid representation to the NMA_CFS [150] approach, but requires K to be pre-fixed. A more advanced fitness function was proposed, which considers a partition's connectedness, compactness, and separability. The proposed method was shown to generally perform better than NMA_CFS [150] on the synthetic datasets. However, NMA_CFS [150] is able to be used when K is unknown, which likely makes it a more useful method in the general case.

A limited amount of work has been published that uses PSO for either the clustering or FS task, and uses a non-EC technique to perform the other task. Marinakis et al. [109] proposed a two-phase approach using PSO for feature selection and a greedy randomised adaptive search procedure (GRASP) for clustering. Their second paper [110] used a multi-swarm version of PSO using a constriction factor in order to improve the feature selection performance compared to the first paper [109]. Kuo et al. [85] used PCA to perform dimensionality reduction before using a hybrid GA-PSO approach to perform clustering for an unknown number of clusters. Another method used the Projection Pursuit (PP) model to reduce dimensionality and then used PSO to perform clustering [183].

Summary

Existing techniques using EC for simultaneous feature selection and clustering [71, 150] clearly show the suitability of EC methods to this task. However, these works have a number of limitations due to their use of variable-length and centroid-based representations, as well as their use of fitness functions that do not account for the interaction between the number of features and clusters.

EC for Subspace Clustering

Subspace clustering can be seen as a form of FS in clustering, as each cluster found lies in a subspace of the data (i.e. a subset of the features are used). Several EC methods have been proposed for performing subspace clustering [130, 167]. However, subspace clustering intrinsically has an even larger search space than normal clustering, as the quantity and choice of features must be made for every cluster, rather than only once for the dataset as in other FS for clustering approaches [128].

2.7.3 GP for Clustering

A small amount of research has been published that uses GP for clustering. A grammar-based method was proposed, which uses GP to create trees containing a series of symbolic logical formulae, where each formula represents a cluster [39]. An instance is assigned to the cluster with the best matching formula, based on the instance's feature values, or to the closest centroid if it matches no formula. Fitness of an individual was measured using a weighted sum of the intra- and inter-cluster distances. While this method does perform some inherent feature selection and construction, the use of only logical predicates and not arithmetic operators may limit the effectiveness of the constructed features. Furthermore, the fitness function does not consider the size of the feature subset used; overly large and specific solutions may be generated on large feature sets.

Boric and Estévez [20] proposed a multi-tree approach where each GP individual contains a tree corresponding to each cluster. An instance is assigned to a cluster by feeding it to each of the trees, and taking the tree with the maximum output as the assigned cluster. This approach performs feature selection and construction as part of the evolutionary process, as the terminal set contains features (only some of which are used), and the function set contains arithmetic operators to combine features. While this approach was shown to be superior to the k-means algorithm, it requires K to be known in advance, so that the number of trees per individual is fixed. This generally means that some domain knowledge is required to use this method, in order to estimate a suitable number of clusters. Furthermore, the authors tested their approach on datasets with a relatively low K (a maximum of $K = 7$). If K was higher, e.g. $K = 40$, it is likely that this approach will perform poorly as training 40 trees simultaneously is very difficult due to a very large search space.

Ahn et al. [5] proposed a more straightforward approach using a simple GP program design where the output of the tree directly corresponds to a

cluster label. The terminal set consists of the feature set and a random constant, and the function set contains the usual arithmetic and other mathematical operators. The output of a tree is rounded using integer rounding to give the cluster label for an instance. This rounding technique can cause issues as it introduces an uneven search space — an output of 0.99 would correspond to the "0" cluster, despite being much closer on the number line to the "1" cluster. Using such a multi-threshold system for cluster assignment is a technique that is known to give poor performance in multi-class classification [105] and it follows this would also be true for clustering. A simple improvement on this would be to use the tree output as an input to another data mining algorithm (e.g. k-means) to give the final cluster assignment.

The Multi-Objective Clustering with Hierarchical Partition Fusions (MCHPF) algorithm [25] uses GP to construct a clustering partition using initial partitions generated by a set of different base clustering techniques (k-means, hierarchical single linkage etc.) These base partitions form the terminal set of the program design, and the function set consists of *consensus* functions, which merge child partitions using the existing Hybrid Bipartite Graph Formulation (HBGF) and Meta-Clustering Algorithm (MCLA) algorithms. The output of the tree is a partitioning of the dataset. The authors used a multi-objective approach whereby both the overall deviation and the connectivity of a partition determines the fitness of an individual. The use of an ensemble method that considers multiple clustering algorithms is a relatively recent development in clustering, and this work is the first to use GP in combination with this technique. MCHPF does not perform any feature selection or construction as the raw features are never used in the GP tree, hence, performance may suffer on large feature sets where the base clustering algorithms begin to perform poorly. Furthermore, the proposed algorithm requires setting a range for the number of clusters, K, as part of the configuration settings. This approach ensures that a reasonable amount of clusters will be generated by the base algorithm, but again

means some domain knowledge is required.

More recently, a GP approach has been proposed based on the idea of novelty search [119], where in lieu of an explicit fitness function, the *uniqueness* (novelty) of a solution in the behavioural landscape[4] is used to determine whether it is used in subsequent generations. This approach was only tested on problems with two clusters, and it is unclear how it would scale as K increases, given that the search space becomes exponentially larger.

Summary

A small number of methods have been proposed that use GP for clustering, including grammar-based, multi-tree, multi-objective, and novelty search-based approaches. However, these methods do not take full advantage of GP's powerful potential for performing FC; these methods perform FC only implicitly as a side-effect of using a GP-based representation. Existing clustering methods, which are well-established in the literature, could benefit significantly from using high-quality constructed features created by a dedicated GP-based FC approach.

2.7.4 PSO for Feature Selection using Feature Grouping

Lane et al. [86] proposed a method using PSO with statistical clustering for FS. They use the standard PSO representation for FS, where each particle has a single dimension for each feature in the feature set. Binary PSO is used, such that a 1 in a given dimension of a particle's position indicates the corresponding feature is selected; a 0 indicates the feature is not selected. After each iteration of the PSO loop, each cluster is analysed and the feature with the highest velocity in the cluster is selected (given a position value of 1), while all the other features have their position values set to 0. Hence, their method ensures that one feature from each cluster is selected. While their method achieved good performance, their

[4]The behaviour of a program considers the program output in conjunction with the context in which the output was produced.

representation is somewhat restrictive: it is not possible to select no features or multiple features from a cluster, which limits performance where all features in a cluster are of poor quality, or when feature interaction occurs. The authors later addressed this limitation by using a Gaussian distribution to allow multiple features to be selected from a single cluster [87]. Their proposed method selects lower numbers of features from a cluster more often than higher numbers (according to a Gaussian distribution), thereby encouraging a small number of features (to maximise dimensionality reduction) while still allowing multiple features where beneficial. While this approach still forces the selection of at least one feature per cluster, their results showed superior performance compared to a standard PSO feature selection algorithm.

Nguyen et al. [122] also proposed a PSO-based method that used statistical clustering for FS. Their method takes a different approach by using a new particle representation that directly encodes the number of features to be selected from each cluster. For each of the K clusters, a maximum of s features are allowed to be selected, where s is defined to be the square root of the cluster size. Each cluster is then allocated s dimensions in the particle representation, where each dimension represents one selected feature. A particle's position encodes a set of candidate selected features. Each position dimension contains a value in the interval $[0, 1]$ where the value represents a certain feature in the corresponding cluster. A series of intervals according to the cluster size are used to map a value to a feature (or to "null", indicating no feature selected). This new representation allowed their proposed method to achieve similar performance compared to existing PSO feature selection techniques, while selecting many fewer features. By using a series of intervals to map position values to features, there will be a sudden change in a particle's fitness when a value is changed across an interval, and no change if a value is changed within the same sub-interval. This produces an uneven fitness landscape, which can affect the training effectiveness of the method. The authors later addressed this limitation

by introducing a Gaussian transformation technique, which introduces a chance that values close to an interval will instead be mapped to the neighbouring sub-interval [121]. This technique produces a smoother fitness landscape, as small changes in a position value will slightly change the probability of a feature being selected. Their results showed general improvements in performance and reduced the number of selected features compared to their base method.

2.8 Summary

This chapter reviewed a number of key concepts that underpin this thesis, including machine learning, clustering, feature manipulation, manifold learning, and evolutionary computation (with a focus on GP and PSO). A comprehensive survey of literature related to the use of EC for feature manipulation in unsupervised learning was also performed. This survey highlighted the potential of EC for performing FM in unsupervised learning tasks, but also outlined a number of significant limitations and gaps in the literature, which, in conjunction with Section 1.2, provide the key motivations of this thesis:

- **EC for feature selection in clustering:**
  The potential for EC-based feature selection to improve clustering algorithm performance has been demonstrated in the literature. However, simultaneous feature selection and clustering — which allows suitable features to be found alongside good clusters — has been performed by only a handful of methods. These methods are limited by their assumptions around the number of clusters; encoding schemes; rudimentary fitness functions; and lack of application to large datasets.

- **EC for feature construction in clustering:**
  The use of GP for directly performing clustering is known to have a number of limitations, including the difficulty in mapping a single GP output to a large number of potential clusters. While GP has

seen some initial use for performing clustering, but has never been used to explicitly perform FC in order to improve the performance of clustering algorithms. The capability of GP for performing FC is well-known in the supervised learning domain, and the lack of its use for clustering represents a significant gap in the field.

- **Creating benchmark feature selection datasets:**
  Synthetic benchmark datasets are often used to evaluate ML algorithms in a controlled manner. While the creation of clustering datasets has been well-studied, there has been minimal investigation into creating synthetic features for benchmarking FS algorithms. Existing techniques primarily duplicate, scale, or add noise to existing features; such techniques are unrealistic and do not challenge modern FS algorithms. GP-based FC could be used to create challenging synthetic features across a range of supervised and unsupervised problems by encouraging complex redundancies between features that are hard for a FS algorithm to detect.

- **Interpretable manifold learning:**
  Manifold learning is one of the latest trends in performing high-quality unsupervised dimensionality reduction, with significant success in tasks such as data visualisation. However, state-of-the-art MaL algorithms are nearly exclusively black-box in nature, giving practitioners little insight into the meaning of the created low-dimensional space or visualisation. Given the use of MaL algorithms for exploratory data analysis, it is essential that they are both high-performing and interpretable. GP is well-regarded for its ability to create interpretable functions that can map a feature space to a new lower-dimensional constructed feature space. Despite this, GP has never been used for directly performing MaL.

Each of the following four chapters focuses on addressing one of these limitations.

Chapter 3

Particle Swarm Optimisation for Simultaneous Feature Selection and Clustering

3.1 Introduction

While EC methods have seen some limited use for simultaneous clustering and feature selection [71, 150], these methods have all used centroid-based encodings. Centroid encodings may produce centroids that are invalid solutions; fail to train effectively due to centroid ordering; and are difficult to use effectively when the number of clusters (K) is not known. Furthermore, the fitness functions used do not adequately consider the interaction between the number of features selected and the number of clusters found. There is a clear need for new representations and refined fitness functions that address these limitations.

3.1.1 Chapter Goals

This chapter aims to investigate a new PSO approach to performing simultaneous feature selection and clustering. The use of new representations and fitness functions is anticipated to improve performance while selecting fewer features than existing approaches. In

particular, this chapter will investigate:

- whether centroid or medoid clustering representations in PSO are most suitable for performing simultaneous feature selection and clustering;
- if the centroid representation can be improved to reduce the likelihood of invalid solutions being created;
- whether a flexible variation of the medoid approach that does not require K to be pre-defined can automatically find a good K while selecting few features; and
- if a multi-stage approach can improve the flexible medoid approach by utilising heuristics for finding K and local search for improving the medoids found.

3.1.2 Chapter Organisation

Sections 3.2 to 3.4 address the first two goals of this chapter by proposing a number of PSO representations for simultaneous FS and clustering. The performance of these representations are compared across a number of datasets. Sections 3.5 to 3.6.1 tackle the third and fourth goals, by extending the medoid representation proposed in Section 3.2 into a multi-stage approach. Different variations of this approach are compared with the initial medoid representation and to other baselines. A summary of the main findings of this chapter is provided in Section 3.8.

3.2 The Proposed PSO Representations

This section proposes PSO methods for both the cases where K is pre-defined and where K is automatically determined by the learning process. We also propose a fitness function that can be used to give good simultaneous feature selection and clustering solutions in both cases. The overall design of the PSO algorithm for performing simultaneous feature

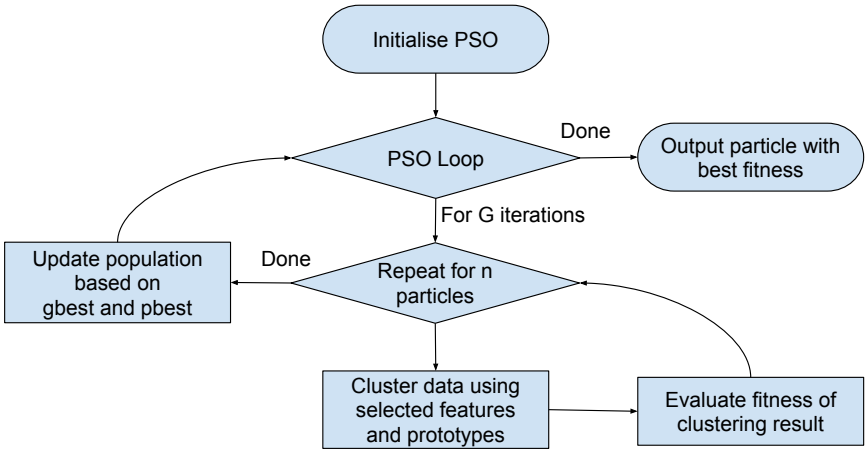


Figure 3.1: The overall flow of the PSO process for all of the proposed representations.

selection and clustering is shown in Figure 3.1. The variations between the different representations occur in the initialisation mechanisms ("Initialise PSO") and in how prototypes are represented ("Cluster data using selected features and prototypes").

3.2.1 Pre-defined K

We use a **centroid** representation (as shown in Figure 3.2) as a base PSO method for performing simultaneous feature selection and clustering. This is the same representation as in the NMA_CFS approach [150], but here we only use it when K is known and hence it has a fixed length of $m + Km$, where the first m dimensions correspond to the m features in the dataset. The position value for a given feature determines if it has been selected by the PSO algorithm. A position value greater than 0 indicates a selected feature; less than 0 indicates the feature is not selected. The remaining Km dimensions correspond to the co-ordinates of the K cluster prototypes. Instances are assigned to the cluster with the closest prototype by Euclidean distance based on the selected features only.

To address the issue where the centroid representation could produce invalid solutions, we propose two seeding techniques, which ensure at

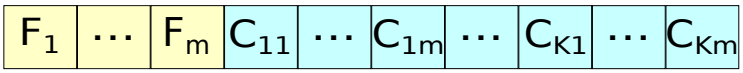


Figure 3.2: Centroid representation for simultaneous clustering and feature selection.

least one of the initial particles in the swarm will contain centroids that produce a valid partition with the correct K. These two techniques are described below:

- The ***k*-means seeded centroid** approach first runs the k-means algorithm on the dataset to give K viable centroids. One particle in the swarm is initialised to contain these K centroids and the remaining particles are initialised as normal (i.e. randomly). For all particles, the FS component of the particle is randomly initialised. This technique ensures at least one viable particle is produced, while also using the result of k-means as a good starting point for the PSO learning process.
- The **dataset seeded centroid** approach initialises the swarm by randomly choosing instances from the dataset to act as initial centroids. Each particle is initialised by randomly choosing K unique instances and using their feature values as the initial positions of the K centroids. This approach still uses a centroid representation; centroids are not constrained to take instance values in subsequent iterations. This approach ensures that all particles in the swarm will initially give valid solutions.

The **medoid** clustering approach can be extended to perform FS by adding a dimension per feature, as shown in Figure 3.3. This representation has a length of $m + n$ for n instances in the dataset, and uses the same FS technique as the centroid approach. Instead of using centroids, each instance is assigned a dimension in the representation, and the position value of an instance's dimension determines if it is a cluster prototype. The instances that correspond to the K highest

position values are chosen to be the K prototypes. As before, instances are assigned to the nearest prototype by Euclidean distance.

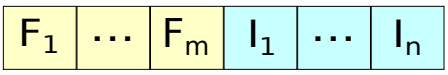


Figure 3.3: Medoid representation for simultaneous clustering and feature selection, where K is known in advance.

3.2.2 No Pre-defined K

As discussed previously, the use of a centroid encoding when K is not pre-defined requires a variable length encoding (or a technique to turn centroids "on" and "off" [46]). The medoid representation can be easily extended to allow for a flexible number of clusters to be generated while maintaining a fixed-length encoding. When K was fixed, the instances with the K highest position values were chosen as medoids. An alternative technique is to choose all instances where the position values are above some threshold Θ. K will then vary based on the number of instances meeting this threshold. The representation for this **dynamic medoid** approach is shown in Figure 3.4. This representation includes a single additional dimension corresponding to Θ. The length of this representation is $m + n + 1$. While Θ could be a constant value (e.g. 0), we suggest that it may be more effective to allow the PSO process to automatically vary Θ, as changing Θ can add or remove multiple clusters from a solution at a time.

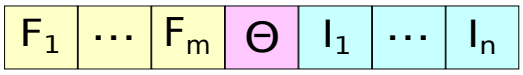


Figure 3.4: Medoid representation for simultaneous clustering and feature selection, where K is **unknown**.

3.2.3 Fitness Function

In order to perform effective clustering and FS, a fitness function should encourage good clustering performance while minimising the number of

features selected. The NMA_CFS [150] algorithm used the J_2 fitness function (shown in Equation (3.1)), which measures clustering performance using the trace scatter metric and applies two penalty terms to minimise the number of features and clusters selected.

$$J2 = trace(S_w^{-1} S_b) \times \frac{m - m^{'}}{m - 1} \times \frac{K_{max} - K}{K - 1} \tag{3.1}$$

where m and $m^{'}$ are the total number of features and the number of **selected** features respectively. K and K_{max} are the number of clusters and the maximum number of clusters respectively. K_{max} is defined as $\sqrt{n}$ (as is common in the literature [126]) where n is the number of instances in the dataset. S_w and S_b are the within- and between-cluster scatter matrices, which measure cluster compactness and separability respectively and are defined as follows:

$$S_w = \frac{1}{n} \sum_{i=1}^{K} \sum_{I_a \in C_i} (I_a - Z_i)(I_a - Z_i)^T \tag{3.2}$$

$$S_b = \sum_{i=1}^{K} \frac{|C_i|}{n} (Z_i - Z^*)(Z_i - Z^*)^T \tag{3.3}$$

where C_i represents the i^{th} cluster, and Z_i and $|C_i|$ are the mean of the i^{th} cluster and the number of instances in the i^{th} cluster respectively. I_a is an instance within cluster C_i. The dataset mean is given by Z^*.

While we base our fitness function on Equation (3.1), we found that a number of adjustments could be made to improve performance. The clusters produced can be improved by punishing outliers more heavily by using the sum of squared Euclidean distances instead of the squared difference to measure intra- and inter-cluster dissimilarity. The number of clusters should not be penalised linearly; we should apply only a small penalty to smaller values of K, but apply an increasingly larger penalty as K approaches K_{max}. Doing so allows a wider range of small potential K values to be considered by the learning algorithm, while still

penalising K values that are so large as to produce overly specific clusters. This change improved the accuracy of K for the dynamic medoid approach. Our proposed fitness function with these improvements is shown in Equation (3.4).

$$\text{Fitness} = \frac{Between_{sum}}{Within_{sum}} \times \frac{m - m^{'}}{m - 1} \times (1 - \frac{\log K}{\log K_{max}}) \tag{3.4}$$

$$Within_{sum} = \frac{1}{n} \sum_{i=1}^{K} \sum_{I_a \in C_i} d(I_a, Z_i)^2 \tag{3.5}$$

$$Between_{sum} = \frac{1}{n} \sum_{i=1}^{K} |C_i| d(Z_i, Z^*)^2 \tag{3.6}$$

where $d(I_a, I_b)$ is the Euclidean distance between instances I_a and I_b, defined in the standard way:

$$d(I_a, I_b) = \sqrt{(I_{a1} - I_{b1})^2 + (I_{a2} - I_{b2})^2 + ... + (I_{am} - I_{bm})^2} \tag{3.7}$$

When computing the Euclidean distance between two instances, we use **all** features, not only the selected ones. If only the selected features are used, the fitness function would become overly biased towards a low K and $m^{'}$; selecting fewer features reduces the amount of information available to the clustering method, reducing the extent to which the dataset can be separated into smaller, more specific clusters. By using all features, we ensure the original structure of the dataset is considered when evaluating the performance of a solution.

3.3 Experiment Design

We evaluated the performance of the PSO representations on a range of datasets using a variety of evaluation metrics. We also compared our results to those produced by the standard k-means when all features are used. As all of the methods are non-deterministic due to their use of random initialisation, we ran each method 30 times and computed the

mean of each metric across all runs. Unfortunately we were unable to compare to the NMA_CFS algorithm as the authors were unable to provide their source code and we were not able to reproduce their results when we re-implemented their approach. The remainder of this section will discuss the datasets, evaluation metrics and PSO parameters used in detail.

3.3.1 Datasets

We tested the PSO representations across a variety of real-world and synthetic datasets, which are summarised in Table 3.1. The real-world datasets are ones that are commonly used in the clustering literature, and all are from the UCI machine learning repository [31]. All of these are classification datasets; it is common practice to use classification datasets for clustering by excluding the class label from the learning process. The class labels are then used to evaluate how well the clusters produced match the known classification. As clustering a classification dataset is harder than clustering a specifically-designed clustering dataset (due to potential heterogeneity in classes), we generally use real-world datasets with small K, but include the Movement Libras dataset (with $K = 15$) to give an indication of performance on many-class classification problems.

The synthetic datasets were generated by Handl et al. [56] and are also widely used. We used a range of synthetic datasets, which contain 10, 50, or 100 features and 10, 20, or 40 clusters. Testing the representations on large m and K shows how they scale as the search space increases. All datasets were standardised so that each feature had zero mean and unit variance to ensure features contribute equally to the clustering process.

3.3.2 Evaluation Metrics

We evaluated the performance of each of the representations using five commonly used metrics. These included two internal metrics, which directly measure the quality of a partition. The first of these is the trace

Table 3.1: Datasets used in the experiments.

Real-World UCI datasets from [31].				Synthetic datasets from [56].			
Name	No. of Features	No. of Instances	No. of Classes	Name	No. of Features	No. of Instances	No. of Clusters
Iris	4	150	3	10d10c	10	2730	10
Wine	13	178	3	10d20c	10	1014	20
Movement Libras	90	360	15	10d40c	10	1938	40
				50d10c	50	2699	10
Breast Cancer	9	683	2	50d20c	50	1255	20
				50d40c	50	2335	40
Image Segmentation	18	683	7	100d10c	100	2893	10
				100d20c	100	1339	20
Dermatology	34	359	6	100d40c	100	2212	40

scatter metric (as defined in Section 3.2.3), which measures the compactness and separability of a partition. In addition, we computed the sum of the intra-cluster distances, which measures compactness directly. This is the metric that the k-means directly optimises.

We also used three external metrics, which measure how well the clustering solution matches the dataset's known classification. The class purity metric measures how homogeneous each cluster is in terms of the class labels of the instances it contains. The error rate (ER) is a metric proposed by Cura [27] that measures how well pairs of instances agree on their class labels and cluster memberships. The true positive rate (TPR) measures the proportion of instances in the same class that are also in the same cluster.

For the sum-intra cluster distance and ER metrics, a smaller value represents a better result; for the other three metrics, higher is better. Each of the five metrics, asides from the scatter trace metric, are defined as follows:

1. Sum intra-cluster distance:

$$\sum \text{Intra} = \sum_{i=1}^{K} \sum_{I_a \in C_i} d(I_a, Z_i) \tag{3.8}$$

2. Class purity: computed according to the following steps:

(3.9)

   (a) For each cluster, find the majority class label of the instances in that cluster.

   (b) Count the number of correctly classified instances, where an instance is correctly classified if it belongs to the majority class.

   (c) Find the class purity as the fraction of correctly classified instances across the whole partition.

3. ER [27]:

$$\text{ER} = \Big[\frac{2}{n(n-1)} \times \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} disagree(I_a, I_b)\Big] \times 100\% \tag{3.10}$$

   where $disagree(I_a, I_b)$ is 0 if instances I_a and I_b are either both in the same classes and same clusters or in different classes and clusters, and 1 otherwise.

4. TPR:

$$\text{TPR} = \frac{1}{totalPairs} \times \sum_{i=1}^{\#Cl} \sum_{I_a, I_b \in Cl_i} agree(I_a, I_b) \tag{3.11}$$

   where Cl_i represents the i^{th} class and $\#Cl$ is the number of classes. $agree(I_a, I_b)$ is 1 if I_a and I_b are in the same cluster, and 0 otherwise. $totalPairs$ is the sum of the number of pairs of instances in each class.

3.3.3 PSO Parameters

Several parameters must be set in the PSO algorithm. We use commonly-used parameters [168]: 100 iterations, a swarm size of 30, $v_{max} = 6$, $w = 0.729844$, and $c_1 = c_2 = 1.49618$. These parameters are constant across all datasets.

3.4 Results and Discussion

The performance of the approaches are shown in Tables 3.2 and 3.3 respectively. KMS-Centroid is the k-means seeded centroid approach, DS-Centroid is the dataset seeded centroid approach, and D-Medoid is the dynamic medoid approach. We provide the mean results for each of the evaluation metrics discussed previously, as well as the mean number of features selected ($m^{'}$) and **non-empty** clusters produced (K). The Σ Intra and ER metrics are marked with a $*$, as a reminder that they should be minimised. The best result for each external metric is bolded, and a method is bolded if it has the best performance on a majority of the external metrics. The results will be analysed for the real-world and synthetic datasets in turn.

3.4.1 Results on Real-World Datasets

Most of the pre-fixed representations perform similarly on the easy Iris dataset, with the centroid approach achieving slightly better results on the external metrics. On the more difficult real-world datasets, the centroid approach begins to struggle; it is not able to generate 15 clusters successfully on the Movement Libras dataset, and it has the worst performance across the metrics on the Image Segmentation and Dermatology datasets. The seeded KMS- DS-centroid approaches are able to outperform the normal centroid approach on these datasets and are always able to correctly generate K viable clusters. The medoid approach performs the best of the pre-fixed approaches as it consistently selects a relatively small number of features while achieving competitive performance on both the internal and external metrics. For example, on the Movement Libras dataset it selects only 20.67 features on average while achieving similar results to both KMS-centroid and k-means, which use 39.57 and 90 features respectively.

k-means outperforms all of the PSO methods on the Wine and Breast Cancer datasets, but performs poorly in comparison on Iris. In general,

Table 3.2: Performance on Real-World Datasets.

Dataset	Method	$m^{'}$	K	Scat.	Σ Intra *	Purity	ER *	TPR
Iris	**Centroid**	1	3	23.57	16.94	**0.9111**	**10.21**	**0.8681**
	KMS-Centroid	1	3	22.93	16.92	0.9036	10.99	0.8612
	DS-Centroid	1	3	22.50	16.91	0.8987	11.48	0.8568
	Medoid	1	3	22.66	16.92	0.9004	11.30	0.8584
	D-Medoid	1.100	3.167	23.83	16.64	0.9049	11.36	0.8308
	k-means	4	3	12.03	16.93	0.7844	19.97	0.7718
Wine	Centroid	2.600	3	11.05	57.70	0.9045	11.79	0.8212
	KMS-Centroid	2.633	3	11.51	57.13	0.9270	9.386	0.8536
	DS-Centroid	2.267	3	10.86	57.34	0.9243	9.755	0.8479
	Medoid	2.400	3	11.18	57.28	0.9232	9.887	0.8458
	D-Medoid	2.733	3	10.11	57.72	0.9077	11.68	0.8237
	***k*-means**	13	3	13.69	56.39	**0.9610**	**5.287**	**0.9152**
Move. Libras	Centroid	45.37	10.63	12.52	336.0	0.2489	26.62	0.4164
	KMS-Centroid	39.57	15	34.80	240.7	0.4506	9.877	0.3704
	DS-Centroid	35.5	15	28.47	259.3	0.4148	11.04	0.3358
	Medoid	20.67	15	36.07	237.8	**0.4680**	9.693	0.3820
	D-Medoid	23.13	6.267	13.89	300.3	0.2799	18.57	**0.4498**
	k-means	90	15	37.24	237.5	0.4640	**9.569**	0.3902
Breast Cancer	Centroid	1.333	2	5.505	142.5	0.9375	11.71	0.9094
	KMS-Centroid	1.367	2	5.310	142.6	0.9366	11.87	0.9081
	DS-Centroid	1.233	2	5.222	142.8	0.9348	12.18	0.9070
	Medoid	1.167	2	5.294	143.0	0.9343	12.27	0.9071
	D-Medoid	1.733	2.733	9.778	137.6	0.9414	13.80	0.8417
	***k*-means**	9	2	6.774	137.8	**0.9587**	**7.925**	**0.9341**
Image Seg.	Centroid	6.300	7	35.28	661.3	0.5556	20.37	0.7226
	KMS-Centroid	4.533	7	51.60	636.5	0.5845	17.07	**0.7444**
	DS-Centroid	4	7	56.69	628.5	0.5857	16.44	0.7124
	Medoid	3.800	7	67.51	611.4	**0.6072**	**14.46**	0.6684
	D-Medoid	4.5	7.300	55.60	612.0	0.5950	15.20	0.6550
	k-means	18	7	50.92	607.1	0.6043	15.30	0.6715
Derm.	Centroid	11.60	6	46.70	183.6	0.7246	15.69	0.7288
	KMS-Centroid	8.833	6	72.87	176.1	0.8024	11.79	0.7788
	DS-Centroid	7.867	6	71.46	175.9	0.8044	10.97	0.8035
	Medoid	6.5	6	86.21	172.9	**0.8318**	**10.83**	0.7554
	D-Medoid	7.867	4.267	61.76	182.7	0.7344	14.84	**0.9088**
	k-means	34	6	92.22	178.0	0.8099	12.26	0.7839

the PSO methods are able to achieve good performance given that they use many fewer features than k-means. All of the methods struggle on the Movement Libras dataset with TPR and purity values under 50%.

This dataset is known to be difficult for clustering algorithms [150] as the class labels do not correspond well to hyper-spherical clusters.

The dynamic medoid approach (D-Medoid) performs reasonably well on the Iris, Wine, Breast Cancer and Image Segmentation datasets, generating K to within 1 of the actual number of classes and achieving similar results on the metrics to the other PSO methods. It struggles to achieve good results on the Movement Libras dataset where it only finds 6.267 clusters on average and has a correspondingly high error rate. It is interesting to note that on both the Movement Libras and Dermatology datasets it has the highest TPR value despite having incorrect K; this suggests it is creating a few big clusters, each of which contain multiple classes. This highlights a fundamental issue in clustering when K is unknown – it can be difficult to consistently determine across varying datasets when one large cluster should be split into two smaller clusters.

In general, the medoid approach is superior to all other pre-fixed approaches on the real-world datasets, including the centroid approaches, which are much more widely used.

3.4.2 Results on Synthetic Datasets

Unlike the real-world datasets, the synthetic datasets are designed specifically for evaluating clustering algorithms. Hence, they may give a better indication of the performance that can be expected on an unlabelled dataset.

As shown in Table 3.3 (and Table 3.3 continued), the basic centroid approach struggles to find K viable clusters on all of the synthetic datasets that contain 20 or 40 clusters. Of the other pre-fixed K approaches, the medoid approach selects the fewest features while achieving the best results across the metrics on all of the synthetic datasets with 50 or 100 features. This approach even outperforms the k-means approach on these datasets despite using many fewer features. This suggests two things: the medoid approach may be able to better

Table 3.3: Performance on Synthetic Datasets.

Dataset	Method	$m^{'}$	K	Scat.	Σ Intra *	Purity ↑	ER *	TPR
10d10c	Centroid	2.800	10	11.79	803.3	0.6448	13.59	0.5608
	KMS-Centroid	4.033	10	14.69	703.3	0.8179	7.271	0.7103
	DS-Centroid	3.333	10	14.34	734.3	0.7713	8.442	0.6918
	Medoid	3.933	10	15.18	708.0	0.8056	7.864	0.6646
	D-Medoid	3.767	7.467	11.83	753.8	0.7458	9.727	0.7379
	***k*-means**	10	10	17.50	643.8	**0.9281**	**3.983**	**0.8001**
10d20c	Centroid	5.100	12.20	9.875	307.1	0.3854	22.84	0.6632
	KMS-Centroid	5.167	20	55.55	159.5	0.8551	2.834	0.7914
	DS-Centroid	4.767	20	45.13	174.0	0.8084	3.349	0.7543
	Medoid	4.967	20	64.74	143.8	**0.9160**	**1.504**	0.8727
	D-Medoid	4.300	12.73	39.17	188.8	0.7527	4.068	**0.9037**
	k-means	10	20	70.37	143.8	0.9018	2.095	0.8306
10d40c	Centroid	5.367	20.80	8.541	585.8	0.3334	16.13	0.6407
	KMS-Centroid	5.300	40	55.08	311.4	0.8388	1.677	0.7411
	DS-Centroid	5.233	40	39.87	362.8	0.7447	2.577	0.6260
	Medoid	5.667	40	56.81	292.8	0.8734	1.305	0.7821
	D-Medoid	3.900	13.10	20.19	502.4	0.4861	6.786	0.8267
	***k*-means**	10	40	73.06	261.2	**0.9188**	**0.9848**	**0.8386**
50d10c	Centroid	22.97	10	23.72	1330	0.5807	23.94	**0.5782**
	KMS-Centroid	17.67	10	56.70	972.9	0.7125	16.82	0.5545
	DS-Centroid	16.60	10	52.43	995.2	0.6948	18.53	0.5524
	Medoid	13.67	10	62.12	938.9	0.7470	14.36	0.5655
	D-Medoid	13.47	18.23	133.5	717.6	**0.8773**	**9.334**	0.4115
	k-means	50	10	70.98	967.0	0.7365	16.06	0.5698
50d20c	Centroid	24.23	12.47	23.83	813.2	0.3314	29.54	**0.6670**
	KMS-Centroid	22.30	20	119.0	480.6	0.6737	11.95	0.5876
	DS-Centroid	20.43	20	99.82	529.8	0.6621	8.742	0.5879
	Medoid	14.17	20	112.3	453.2	**0.7459**	**7.216**	0.6203
	D-Medoid	15.33	17.40	97.53	486.5	0.7115	10.16	0.6655
	k-means	50	20	136.0	477.6	0.6895	11.72	0.6021
50d40c	Centroid	26.77	22.90	29.76	1520	0.2905	24.19	0.6073
	KMS-Centroid	22.87	40	176.2	852.1	0.6696	8.309	0.5551
	DS-Centroid	19.47	40	131.6	1001	0.6238	5.387	0.5272
	Medoid	14.37	40	156.9	860.9	**0.7053**	**4.135**	0.5687
	D-Medoid	14.53	22.80	84.56	1156	0.5042	11.58	**0.6274**
	k-means	50	40	190.7	847.2	0.6786	9.822	0.5825

explore the search space than the centroid approaches as it can more effectively utilise the social knowledge of the swarm, and secondly, that k-means fails to scale well for large k or m, whereas PSO is able to scale

Table 3.3: Performance on Synthetic Datasets (continued).

Dataset	Method	$m^{'}$	K	Scat.	Σ Intra *	Purity	ER *	TPR
100d10c	Centroid	47.5	10	29.04	2099	0.5883	22.00	0.5954
	KMS-Centroid	39.53	10	65.85	1533	0.7392	14.04	0.6016
	DS-Centroid	39.90	10	63.18	1572	0.7230	15.51	0.6035
	Medoid	33.53	10	71.45	1470	0.7808	11.36	**0.6488**
	D-Medoid	34.43	19.07	193.3	1060	**0.9092**	**8.459**	0.4302
	k-means	100	10	94.17	1592	0.7600	13.76	0.6217
100d20c	Centroid	49.80	12.93	27.24	1222	0.3239	30.39	0.6311
	KMS-Centroid	46.90	20	149.8	740.1	0.6709	11.42	0.5933
	DS-Centroid	45	20	138.8	777.4	0.6514	11.31	0.5852
	Medoid	32.17	20	157.2	671.2	**0.7610**	**7.653**	0.6200
	D-Medoid	35.63	18.5	146.3	711.8	0.7268	9.342	**0.6414**
	k-means	100	20	186.2	733.7	0.6921	12.48	0.6184
100d40c	Centroid	49.53	22.77	39.72	2052	0.2868	24.05	0.6019
	KMS-Centroid	47.40	40	263.0	1152	0.6742	8.729	0.5618
	DS-Centroid	43.60	40	217.9	1277	0.6625	6.074	0.5668
	Medoid	32.73	40	246.5	1107	**0.7382**	**4.398**	0.5952
	D-Medoid	34.97	22.93	136.3	1439	0.5516	14.14	**0.6705**
	k-means	100	40	301.7	1148	0.6874	10.26	0.5903

effectively. On the datasets with 10 features, the KMS approach can also perform well relative to the medoid approach. k-means is also able to perform well — it has the lowest error rate and highest purity on 10d10c and 10d40c, as well as the best results for the internal metrics. These datasets have a small number of features and so k-means does not struggle as it does on the other synthetic datasets. Each of the two seeded centroid approaches perform better on different datasets, with the DS approach generally performing more effectively on the larger datasets.

The dynamic medoid approach is inaccurate in terms of the mean number of clusters it produces on all synthetic datasets asides from 10d10c, 50d20c and 100d20c. On the datasets where it overestimates K (50d10c and 100d10c), it achieves the best scatter, Σ intra, purity and ER results despite obviously performing badly. This highlights the difficulty in measuring good clustering performance and in designing a good fitness function – if we used only scatter fitness as our fitness function, we would likely produce many more clusters than is correct. The purity

and ER metrics are also biased towards large K. As the number of clusters increases, the homogeneity of any given cluster is likely to be higher as clusters become smaller and more specific. This leads to an increase in purity. In a similar vein, a higher K also tends to decrease the ER: the more clusters present, the more likely it is that instances from different classes will be assigned to different clusters. As the majority of pairs of instances in a dataset will belong to two different classes when there are more than two classes, the number of disagreements between instance pairs will tend to decrease as K is increased. The TPR metric does not have this issue as it considers only the true positives relative to the number of actual positives; the dynamic medoid approach achieves a low TPR value on 50d10c and 100d10c. These issues demonstrate the need for care when evaluating clustering algorithms, as the number of clusters may have an unexpected effect on how metrics behave.

While the dynamic medoid approach performs poorly compared to the pre-fixed approaches, the case where K is unknown is a much more difficult problem [46]. We believe that the success of the pre-fixed medoid approach and the fact that the dynamic approach can use a fixed-length representation suggest it is worth exploring this direction further.

3.5 A Multi-Stage Approach

While the dynamic medoid approach struggled on the synthetic datasets, it clearly has significant potential given that the fixed medoid approach outperformed the centroid approaches across the datasets. The key limitations of the dynamic medoid approach were its inaccurate determination of K (particularly on synthetic datasets); its tendency to select fewer features at the expense of clustering performance; and its inability to fine-tune cluster centres due to the restrictions given by a medoid representation. To address these, this section proposes a multi-stage approach that provides a number of improvements to the dynamic medoid approach.

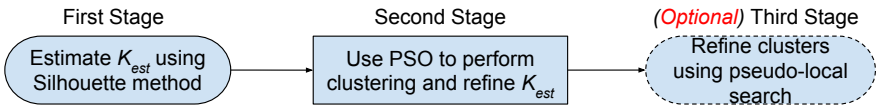


Figure 3.5: The overall flow of the multi-stage approach.

The overall design of the three stages is shown in Figure 3.5. In the first stage, an estimate of K, called K_{est}, is determined using a statistical measure. The second stage then performs simultaneous clustering and feature selection, while using K_{est} as a guide for finding K. K is still dynamic and so can be optimised by the evolutionary search, but individuals that have a K that varies too far from K_{est} will have their fitness punished correspondingly. As methods used to generate K_{est} in the first stage may not give perfect estimates, allowing minor variations to K_{est} allows the EC method to fine-tune the K value. The (optional) third stage then performs a pseudo-local search using a centroid representation to fine-tune the solution produced by the second stage.

The following subsections discuss the design of each of the stages in detail.

3.5.1 First Stage

The Silhouette method described in Section 2.2.4 was used in this study to produce K_{est} in the first stage as it was empirically found to be the most accurate method tested. k-means was used to cluster the data in the first stage for each potential k in the range $[2, \sqrt{n}]$, as suggested in [126], and then the average silhouette for each K was computed. The K with the highest average silhouette is chosen as K_{est}. The silhouette method is non-deterministic and produces a large variation in K_{est} values across different runs. To address this we run the algorithm 30 times and take the median K_{est} to reduce variation, producing more consistent K_{est} values.

3.5.2 Second Stage

The output of the first stage is a single K_{est} value. The second stage uses this value as a heuristic to guide the search by PSO for the number of

clusters. We use the medoid representation introduced in Section 3.2.2.

The fitness function introduced in Section 3.2.3 balanced three key criteria required to measure the quality of a given PSO solution: the clustering performance, the number of features used, and how large K was, which can be formulated as follows:

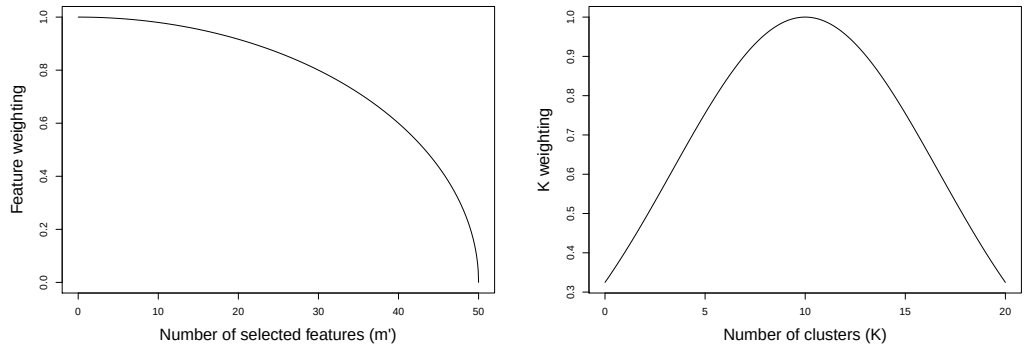
$$\text{Fitness} = \text{Cluster Performance} \times \text{Feature Weighting} \times K \text{ Weighting} \quad (3.12)$$

The first component, measuring the cluster quality, is unchanged. To further address the bias between the number of features selected and cluster performance, we investigate an alternative method for weighting the number of features. The multi-stage approach uses a heuristic K_{est}, and so it is necessary to adapt the third component of this fitness function.

The changes to the second and third components of the fitness function are discussed below.

Feature Weighting:

The most common metric for measuring the goodness of a feature subset is to apply a weighting based on the number of features selected. This is usually expressed as a simple fraction in the form $\frac{m-m'}{m}$ for m total features and m' selected features. Such a weighting applies a linear penalty to the fitness of a given particle with respect to the percentage of features selected. An issue with this approach is that the search process will tend to over-emphasise minimising m' at the cost of cluster performance — it is usually "easier" to improve fitness by reducing m' than by improving cluster performance. Furthermore, the goal of feature selection is generally to reduce the number of features used to an acceptable level; the user may not differentiate between 5% or 10% features being selected as both values of m' are acceptably small. Hence, a linear weighting mechanism may not be ideal; we would like to apply little penalty when m' is reasonably small and then apply an increasing



(a) Elliptical fitness weighting for a dataset containing 50 features.

(b) K fitness weighting for a dataset with K_{est} of 10 using a std. dev. of $\frac{K_{est}}{1.5}$.

Figure 3.6: Fitness weightings for balancing the number of features and clusters.

amount of penalty as m' increases. To achieve this we propose using an elliptical function as shown in Figure 3.6a.

The equation used to compute the elliptical feature weighting is:

$$\text{Feature weighting} = \frac{1}{m}\sqrt{m^2 - (m')^2} \tag{3.13}$$

This formulation is chosen as it produces very little penalty when a relatively small number of features is selected, while still providing some motivation for the PSO process to select as few features as possible without sacrificing clustering performance. As the number of features increases, this function penalises the fitness at an increasing rate until the whole feature set is used and the fitness becomes zero. This formulation better captures the aim of FS: it uses as few features as possible while not overly affecting clustering performance. We trial using both this method and the normal linear method for the feature weighting component of the fitness function in our experiments.

K Weighting:

The final decision required is how to penalise particles that have a K value varying significantly from the K_{est} heuristic. As K_{est} is not a perfect

estimate, we allow small variations from it without any significant penalty. As the variations increase, we should penalise at a higher rate. The use of a Gaussian function was found to give a good balance of these two objectives. Figure 3.6b shows an example of a Gaussian function with $\mu = K_{est}$ and $\sigma = \frac{K_{est}}{1.5}$ where the output is scaled to give 1 (no penalty) when $K = K_{est}$. As shown, the fitness weighting is small for K values between 8 and 12 or so, but becomes large when K is five or 15. The standard deviation must be a function of K_{est} to ensure the function scales effectively; the denominator of 1.5 was chosen empirically. The use of different denominators (e.g. 1 or 2) will increase or decrease the rate of penalty as K varies from K_{est}. We use this Gaussian function as the third component of the fitness function in Equation (3.12).

3.5.3 Third Stage (pseudo-local search)

One key limitation of a medoid-based representation is that cluster prototypes are restricted to the instances in the dataset. It is possible that better clusters may be formed using cluster prototypes that lie elsewhere in the feature space (e.g. halfway between two instances). To address this limitation, while still maintaining the benefits of a medoid approach, we propose the use of a third-stage where the medoid representation is converted to a centroid representation and then the centroids are fine-tuned using another PSO search process. We call this procedure a "pseudo-local search", as particles are initialised to the best solution found in the second stage, but are allowed to explore the search space freely.

Figure 3.7 shows the representation used in the third stage. Each of the K medoids in the best solution from the second stage is used to initialise the position of the particles in the third stage — each medoid is converted to

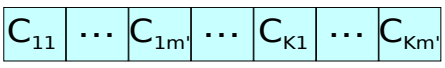


Figure 3.7: Centroid representation used in the third stage.

a centroid with length equal to the number of selected features (m') where the centroid contains the feature values for each of the selected features. Each particle's velocity is randomly initialised. Hence, particles will initially spread out in different directions from the second stage's $gbest$ before beginning to converge again. It is hoped this will allow fine-tuning of the cluster centres, while focusing the majority of the search in an area that is known to give good performance.

3.6 Experiment Design: Multi-Stage Approach

To evaluate the performance of the proposed multi-stage approach, a number of methods were tested across the real-world and synthetic datasets (see Table 3.1). The methods are:

1. 2-Stage Linear: the proposed method using **linear** feature weighting.
2. 2-Stage Elliptical: the proposed method using **elliptical** feature weighting.
3. 3-Stage: the 2-Stage Elliptical method plus the third stage (pseudo-local search) for refining the solutions.
4. k_{est}-means: the standard k-means algorithm but using $K = K_{est}$ as computed by the first stage of the proposed approach. This algorithm is used to evaluate how well the proposed approach is able to refine K based on the heuristic and how well it can perform feature selection.
5. k-means: the standard k-means algorithm, initialised with centroids drawing from instances in the dataset. Note that K is **known**, and so this algorithm is being run on a much easier task.
6. D-Medoid: The proposed single-stage dynamic medoid approach.

All methods have stochastic behaviour and so were run 30 times on each dataset for 500 iterations to ensure search convergence. The PSO methods

(including the pseudo-local PSO) had a swarm size of 100 and used standard PSO parameters [168]: $w = 0.729844$, $C_1 = C_2 = 1.49618$, and velocity clamped between -6 and 6. A fully connected PSO topology was used; *gbest* after 500 iterations gives the best solution. Feature values were scaled linearly to $[0, 1]$.

3.6.1 Evaluation Metrics

As previously, we evaluated our proposed multi-stage methods using the Scatter trace and Sum intra-cluster distance internal metrics (as defined in Section 3.3.2). In place of the three external metrics used in Section 3.3.2, here we used the F-measure (F-m) as the single external metric, which measures how well pairs of instances agree on their class labels and cluster memberships. The F-measure does not suffer from the same biases that were discovered with some of the external metrics earlier in this chapter. The F-measure is defined in Equation (2.15).

3.7 Results and Discussion: Multi-Stage Approach

The results of the experiments are shown in Tables 3.4 and 3.5 for the real-world and synthetic datasets respectively. Each table shows the **mean** number of features selected and clusters produced by each method (note that these are constant for k-means) as well as the method's average performance according to the evaluation metrics. The $\sum$ *Intra* metric is the only one that should be minimised — it is labelled with a $*$ to indicate this. For each of the proposed methods, each result is labelled with a "$+$" or a "$-$" if it is significantly better or worse than the k-means baseline according to a Student's t-test performed with a 95% confidence interval. A lack of a "$+$" or "$-$" indicates no significant difference was found. A label of $\uparrow$ or $\downarrow$ indicates a result is significantly better or worse than the existing D-Medoid method according to the same test. The D-Medoid results vary slightly from those in Section 3.4 as feature scaling rather than standardisation was used. The results are analysed on the real-world and synthetic datasets separately in the following subsections,

and then some general trends are discussed.

3.7.1 Results on Real-World Datasets

Our proposed methods are competitive with k-means on the external metrics and often have superior performance on the internal metrics for the first four real-world datasets, while using a much smaller number of features. The methods also generally outperform D-Medoid on the external metrics on four of the six datasets. The proposed methods perform significantly worse than k-means and D-Medoid across all metrics on the Image Segmentation dataset due to incorrectly choosing $K = 3$. As the value of K_{est} is two on average, PSO is only able to vary K to be 3 without fitness being overly affected. On the Dermatology dataset, the proposed methods achieve a significantly better F-measure value compared to k-means and D-Medoid, despite incorrectly estimating K. This is likely due to the estimated K allowing better-formed clusters; on real-world data, class labels are produced by a human expert and may not correspond well to hyper-spherical clusters.

Another important consideration is that a clustering partition that differs from the known classification is not necessarily a "wrong" clustering — there are many ways to group a dataset based on different characteristics (i.e. feature subsets). Hence, it may be better to consider the performance in terms of the internal metrics as a better measurement of how well the proposed approaches performs In this regard, it is clear that the proposed approaches are able to achieve similar to or better results than k-means on the Iris, Wine, Movement Libras, and Breast Cancer datasets, while using a much smaller number of features. On the other two datasets, the performance is far superior to the k_{est}-means method, while again using a small number of features.

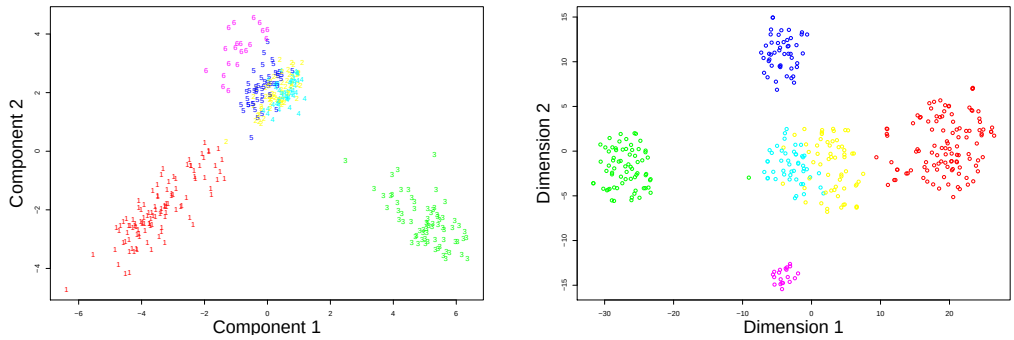
Further Investigation

To analyse why K was being inaccurately estimated on the Image Segmentation and Dermatology datasets, we visualised these datasets

Table 3.4: Real-world datasets.

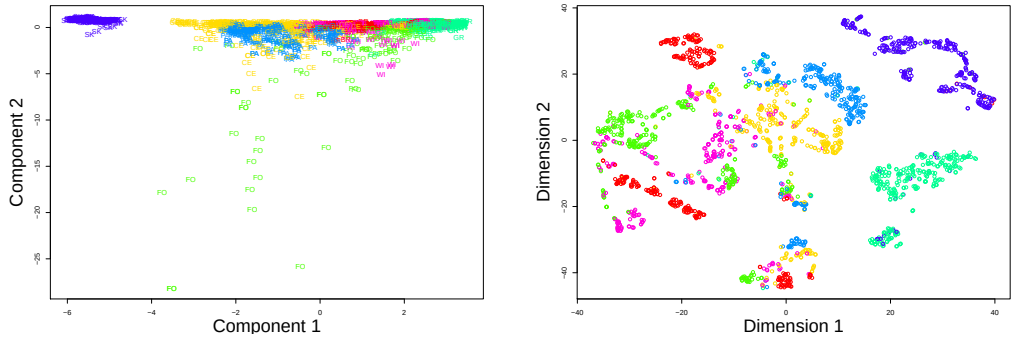
Dataset	Method	m'	K	Scatter	$\sum$ Intra *	Purity	F-m
Iris	2-Stage Linear	1	3	$26.78^{+\downarrow}$	$29.99^{\downarrow}$	$0.9436^{+\uparrow}$	$0.8962^{+\uparrow}$
	2-Stage Elliptical	1	3	$27.4^{+\downarrow}$	$29.96^{\downarrow}$	$0.9493^{+\uparrow}$	$0.9055^{+\uparrow}$
	3-Stage	1	3	$26.9^{+\downarrow}$	$29.99^{\downarrow}$	$0.9449^{+\uparrow}$	$0.898^{+\uparrow}$
	k_{est}-means	4	2	9.8	37.23	0.6667	0.7462
	k-means	4	3	16.37	30.58	0.8404	0.7751
	D-Medoid	1	3.6	35.03	28.29	0.9191	0.8229
Wine	2-Stage Linear	2.27	3.37	$12.53^{\uparrow}$	$89.28^{-\uparrow}$	0.9161^{-}	$0.8152^{-\downarrow}$
	2-Stage Elliptical	3.57	3.4	$13.96^{+\uparrow}$	$88.14^{+\uparrow}$	$0.9418^{\uparrow}$	0.8523^{-}
	3-Stage	3.67	3.4	$14.55^{+\uparrow}$	$87.67^{+\uparrow}$	$0.9541^{+\uparrow}$	$0.8749^{-\uparrow}$
	k_{est}-means	13	2	4.92	104.2	0.6073	0.6357
	k-means	13	3	12.68	88.75	0.9464	0.8947
	D-Medoid	2.27	3	11.14	90.26	0.9167	0.8414
Move. Libras	2-Stage Linear	14.5	17.77	$45.62^{+\uparrow}$	$388.1^{+\uparrow}$	$0.5017^{+\uparrow}$	$0.3423^{\uparrow}$
	2-Stage Elliptical	26.7	17.9	$47.14^{+\uparrow}$	$383.9^{+\uparrow}$	$0.5005^{+\uparrow}$	$0.3501^{\uparrow}$
	3-Stage	26.5	17.73	$49.01^{+\uparrow}$	$380.0^{+\uparrow}$	$0.5012^{+\uparrow}$	$0.3596^{+\uparrow}$
	k_{est}-means	90	12.27	31.28	434.6	0.4226	0.3278
	k-means	90	15	39.01	409.4	0.4705	0.347
	D-Medoid	12.5	6.13	15.21	515.6	0.2859	0.2518
Breast Cancer	2-Stage Linear	1.53	2	$6.062^{-\downarrow}$	$344.2^{-\downarrow}$	0.9407^{-}	$0.8994^{-\uparrow}$
	2-Stage Elliptical	2.5	2	$7.547^{-\downarrow}$	335.6^{-}	$0.9571^{-\uparrow}$	$0.9251^{-\uparrow}$
	3-Stage	2.43	2	$7.807^{-\downarrow}$	334.7^{-}	$0.9573^{-\uparrow}$	$0.9254^{-\uparrow}$
	k_{est}-means	9	2	8.2	332.0	0.9609	0.9313
	k-means	9	2	8.211	332.0	0.9611	0.9316
	D-Medoid	1.6	2.7	10.17	331.0	0.9441	0.8744
Image Seg.	2-Stage Linear	1.33	3	$19.33^{-\downarrow}$	$1245.0^{-\downarrow}$	$0.4251^{-\downarrow}$	$0.4536^{-\downarrow}$
	2-Stage Elliptical	2.13	3	$19.18^{-\downarrow}$	$1242.0^{-\downarrow}$	$0.4275^{-\downarrow}$	$0.4583^{-\downarrow}$
	3-Stage	1.8	3	$19.36^{-\downarrow}$	$1241.0^{-\downarrow}$	$0.4276^{-\downarrow}$	$0.4595^{-\downarrow}$
	k_{est}-means	18	2	4.063	1482.0	0.2857	0.3362
	k-means	18	7	60.86	898.3	0.6426	0.5583
	D-Medoid	2.23	5.23	63.53	984.4	0.6089	0.5725
Derm.	2-Stage Linear	4.53	3.97	68.03^{-}	405.5^{-}	$0.7837^{-\uparrow}$	$0.7886^{+\uparrow}$
	2-Stage Elliptical	7.13	4	$79.26^{-\uparrow}$	$401.6^{-\uparrow}$	$0.8025^{\uparrow}$	$0.8206^{+\uparrow}$
	3-Stage	7.07	4	$81.22^{-\uparrow}$	$400.9^{-\uparrow}$	$0.8083^{\uparrow}$	$0.8311^{+\uparrow}$
	k_{est}-means	34	2.73	55.69	457.3	0.612	0.6113
	k-means	34	6	93.58	387.6	0.8278	0.7351
	D-Medoid	4.37	3.8	68.77	409.4	0.7602	0.7587

using the principal component analysis (PCA) and t-distributed stochastic neighbour embedding (t-SNE) visualisation methods, as shown in Figures 3.8 and 3.9. Each sub-figure shows the results of applying one of these methods to one of the datasets, where the colours represent the class labels of the dataset.



(a) PCA. (b) t-SNE.

Figure 3.8: Visualisations of Dermatology dataset ($K = 6$).



(a) PCA. (b) t-SNE.

Figure 3.9: Visualisations of Image Segmentation dataset ($K = 18$).

On the Dermatology dataset (which has six classes), PCA clearly separates the red and green classes into two distinct clusters. The remaining classes appear as one tightly packed cluster, with the pink class potentially a fourth cluster. This gives three to four clusters on this dataset, consistent with the four average clusters found by the proposed methods. t-SNE more clearly separates the classes into clusters, but there is still overlap between the teal and yellow classes, giving five distinct clusters.

The visualisations on the Image Segmentation dataset are much more unclear; PCA produces a poor visualisation, with only the purple class being clearly separated. t-SNE is clearer — the aquamarine and purple classes are separated well, but the remaining classes all have a fair

amount of overlap. This suggests why the proposed methods find three clusters — two of the classes fit into two clusters well, and the remaining instances have sufficient overlap to produce a single cluster.

In summary, both the linear and elliptical two-stage methods successfully select a small m' on all six of the real-world datasets. The elliptical method has slightly better performance than the linear method while selecting additional features. If the minimum number of features is desired while achieving good performance, then the linear method is best; however, if better performance is preferred at the cost of slightly higher complexity, the elliptical method should be used. The 3-stage method has slightly higher performance than the 2-stage elliptical method across all metrics, which shows the pseudo-local search is able to further refine the solutions.

3.7.2 Results on Synthetic Datasets

Unlike the real-world datasets, the synthetic datasets have classes that map well to hyper-spherical clusters. Thus the external metrics are useful for measuring the performance of the proposed approach, which clearly outperforms k-means and k_{est}-means on all of the synthetic datasets (except for 10d10c) while achieving a low m', especially on the datasets with high m. The proposed approach scales to large datasets more effectively than the k-means algorithm, despite not performing only clustering, but also feature selection and determining K in the same search process. It also performs better than k-means on the internal metrics across the 50d and 100d datasets, where it selects the most useful features to improve clustering.

Despite performing well, the proposed methods are inaccurate in predicting K on several of the synthetic datasets, such as 50d20c and 100d20c where they select $35 - 36$ clusters instead of 20. However, compared to the previously proposed D-Medoid method, the proposed methods predict K more accurately on seven of the nine datasets. The D-Medoid method predicts values of K close to 20 on all the 50d and

Table 3.5: Synthetic datasets.

Dataset	Method	m'	K	Scatter	$\sum$ Intra *	Purity	F-m
10d10c	2-Stage Linear	3.63	8.97	$15.04^{-\uparrow}$	$782.6^{-\uparrow}$	$0.8051^{-\uparrow}$	0.763^{-}
	2-Stage Elliptical	4.93	9.57	$15.63^{-\uparrow}$	$750.0^{-\uparrow}$	$0.8604^{-\uparrow}$	$0.8196^{\uparrow}$
	3-Stage	4.9	9.07	$16.11^{-\uparrow}$	$739.9^{-\uparrow}$	$0.8872^{-\uparrow}$	$0.8678^{+\uparrow}$
	k_{est}-means	10	5.43	11.49	815.9	0.7743	0.7786
	k-means	10	10	17.7	715.7	0.9175	0.833
	D-Medoid	3.3	6.97	12.63	817.7	0.7718	0.7481
10d20c	2-Stage Linear	5.07	20.1	$75.32^{+\uparrow}$	$226.9^{+\uparrow}$	$0.9587^{+\uparrow}$	$0.9408^{+\uparrow}$
	2-Stage Elliptical	6.1	20.17	$85.2^{+\uparrow}$	$216.8^{+\uparrow}$	$0.9828^{+\uparrow}$	$0.9721^{+\uparrow}$
	3-Stage	6.07	20	$90.17^{+\uparrow}$	$213.3^{+\uparrow}$	$0.9953^{+\uparrow}$	$0.9928^{+\uparrow}$
	k_{est}-means	10	15.13	53.31	292.7	0.7907	0.7651
	k-means	10	20	70.02	248.5	0.8887	0.8218
	D-Medoid	4.4	14.7	51.65	282.7	0.8249	0.8305
10d40c	2-Stage Linear	5.47	39.73	$67.26^{-\uparrow}$	$452.0^{-\uparrow}$	$0.9182^{\uparrow}$	$0.8699^{\uparrow}$
	2-Stage Elliptical	6.87	39.7	$78.29^{+\uparrow}$	$417.3^{+\uparrow}$	$0.9615^{+\uparrow}$	$0.9437^{+\uparrow}$
	3-Stage	6.87	40.1	$85.66^{+\uparrow}$	$402.9^{+\uparrow}$	$0.9824^{+\uparrow}$	$0.97^{+\uparrow}$
	k_{est}-means	10	29.83	55.58	499.9	0.8385	0.8234
	k-means	10	40	74.5	433.7	0.9219	0.8657
	D-Medoid	3.83	15.6	27.08	756.4	0.5692	0.5477
50d10c	2-Stage Linear	9.3	13.5	$93.48^{+\downarrow}$	$1072.0^{+\downarrow}$	$0.8191^{+\downarrow}$	0.5197^{+}
	2-Stage Elliptical	14.87	13.23	$96.76^{+\downarrow}$	$1071.0^{+\downarrow}$	$0.8174^{+\downarrow}$	0.5172^{+}
	3-Stage	14.23	13.5	$104.5^{+\downarrow}$	$1045.0^{+\downarrow}$	$0.8152^{+\downarrow}$	0.5125^{+}
	k_{est}-means	50	11.53	88.84	1242.0	0.7679	0.4978
	k-means	50	10	72.87	1306.0	0.7426	0.4865
	D-Medoid	9.07	18.5	144.2	930.6	0.8824	0.5196
50d20c	2-Stage Linear	10.87	34.63	$250.2^{+\uparrow}$	$372.4^{+\uparrow}$	$0.8622^{+\uparrow}$	$0.525^{+\uparrow}$
	2-Stage Elliptical	17.43	34.67	$261.8^{+\uparrow}$	$366.1^{+\uparrow}$	$0.8692^{+\uparrow}$	$0.5171^{+\uparrow}$
	3-Stage	17.33	34.6	$283.1^{+\uparrow}$	$356.8^{+\uparrow}$	$0.858^{+\uparrow}$	$0.4835^{+\uparrow}$
	k_{est}-means	50	28.8	211.3	432.5	0.8057	0.4713
	k-means	50	20	137.5	548.6	0.6858	0.3581
	D-Medoid	10.63	17.43	99.65	539.6	0.7165	0.4167
50d40c	2-Stage Linear	13.57	48	$200.4^{+\uparrow}$	$756.9^{+\uparrow}$	$0.7724^{+\uparrow}$	$0.4888^{+\uparrow}$
	2-Stage Elliptical	19.87	48	$211.7^{+\uparrow}$	$738.8^{+\uparrow}$	$0.788^{+\uparrow}$	$0.4949^{+\uparrow}$
	3-Stage	20.43	48	$230.3^{+\uparrow}$	$708.5^{+\uparrow}$	$0.761^{+\uparrow}$	$0.3669^{+\uparrow}$
	k_{est}-means	50	44.7	214.5	822.6	0.7099	0.2841
	k-means	50	40	192.4	871.9	0.6749	0.2586
	D-Medoid	11.33	26.03	104.5	1062.0	0.5546	0.2455

100d datasets despite K actually varying from 20 to 40. This suggests that D-Medoid cannot search for the true K value as effectively as the proposed methods. It is interesting to note that the K values produced by the proposed methods are always higher than the K_{est} generated by the first stage. This suggests that the fitness function encourages a higher

Table 3.5: Synthetic Datasets (continued).

Dataset	Method	m'	K	Scatter	$\sum$ Intra *	Purity	F-m
100d10c	2-Stage Linear	20	15.87	$148.3^{+\downarrow}$	$1401.0^{+\downarrow}$	$0.8655^{+\downarrow}$	0.5648^{+}
	2-Stage Elliptical	29.4	15.77	$150.9^{+\downarrow}$	$1395.0^{+\downarrow}$	$0.8676^{+\downarrow}$	$0.5707^{+\uparrow}$
	3-Stage	28.9	15.73	$153.4^{+\downarrow}$	$1376.0^{+\downarrow}$	$0.8567^{+\downarrow}$	0.5557^{+}
	k_{est}-means	100	11.27	115.2	1807.0	0.794	0.5623
	k-means	100	10	103.5	2036.0	0.7436	0.5194
	D-Medoid	17.3	18.97	206.1	1287.0	0.9137	0.5549
100d20c	2-Stage Linear	21.9	36	$358.9^{+\uparrow}$	$559.9^{+\uparrow}$	$0.8945^{+\uparrow}$	$0.5627^{+\uparrow}$
	2-Stage Elliptical	33.4	36	$381.9^{+\uparrow}$	$548.2^{+\uparrow}$	$0.8943^{+\uparrow}$	$0.5466^{+\uparrow}$
	3-Stage	34.17	35.93	$399.9^{+\uparrow}$	$536.1^{+\uparrow}$	$0.8858^{+\uparrow}$	$0.5256^{+\uparrow}$
	k_{est}-means	100	26.2	260.4	707.5	0.7865	0.4505
	k-means	100	20	188.1	841.0	0.7011	0.3799
	D-Medoid	19.53	20.23	162.7	748.6	0.7568	0.4435
100d40c	2-Stage Linear	27.37	47	$304.4^{\uparrow}$	$991.7^{+\uparrow}$	$0.7968^{+\uparrow}$	$0.5116^{+\uparrow}$
	2-Stage Elliptical	39.07	47	$315.8^{+\uparrow}$	$980.4^{+\uparrow}$	$0.8057^{+\uparrow}$	$0.5012^{+\uparrow}$
	3-Stage	37.9	47	$338.6^{+\uparrow}$	$940.5^{+\uparrow}$	$0.7624^{+\uparrow}$	$0.3305^{+\uparrow}$
	k_{est}-means	100	42.87	332.0	1146.0	0.7073	0.2837
	k-means	100	40	301.4	1176.0	0.6923	0.2689
	D-Medoid	22.47	23	142.7	1442.0	0.5426	0.1997

number of clusters, perhaps due to the clustering performance metric used. This behaviour is useful in some cases, where K_{est} is below the actual K (e.g. 10d10c, 10d20c and 10d40c), but on the other synthetic datasets where $K_{est} > K_{actual}$, it means that the proposed methods are not able to correctly lower the K found. It would be useful to investigate changing the clustering performance metric in the fitness function to encourage searching values on both sides of K_{est}.

3.7.3 Further Analysis

The two variations of the two-stage approach perform better on different datasets: the elliptical method is best on datasets with small feature sets (Wine, Breast Cancer, Dermatology, 10d) where the linear method selects fewer features at the expense of cluster quality. On the larger feature sets, the elliptical method selects extra features without increasing performance. This is due to the ellipse used: on larger feature sets, e.g. the 50 features seen in Figure 3.6a, the feature weighting is close to 1 for m between 0 and 10, and above 0.9 even when 20 features are selected —

indeed, on the synthetic datasets with 50 features, this method selects 15 to 20 features. On smaller feature sets, this effect is less noticeable, as only a few features are able to be selected before the feature weighting decreases significantly. Investigating a way of dynamically altering the shape of the ellipse used based on the size of the feature set would ensure that the weighting "drop-off" begins earlier on bigger datasets. The two-stage methods have similar values of K across all datasets, indicating that neither is being overly affected by the correlation between m' and K. If this had occurred, the elliptical method would have higher K on the datasets where it selected more features than the linear method.

The three-stage approach is an improvement compared to the two-stage elliptical approach on the internal metrics across all of the datasets (and especially on the hardest synthetic ones). This suggests that fine-tuning the solutions produced with a pseudo-local search is effective, increasing the cluster quality. However, the results on the external metrics are much worse for the three-stage approach on the 50d40c and 100d40c synthetic datasets, contrary to that of the internal metrics. It is not obvious why this occurs — one explanation is that the noise present in the synthetic datasets has a significant effect when K is large (i.e. $K = 40$); as a centroid can take any possible co-ordinates (unlike a medoid), it may be much more sensitive to the noise in the dataset, producing overly specific clusters.

3.8 Chapter Conclusions

This chapter compared a number of different PSO representations for simultaneously performing clustering and feature selection. We proposed an extension to the medoid representation ("D-Medoid") that enables it to be used in the case where the number of clusters is unknown. An improved version of the NMA_CFS fitness function [150] was also introduced. A comprehensive quantitative comparison of the different representations was conducted across a number of real-world and synthetic datasets with a range of different characteristics. It was found that a medoid representation could achieve the best performance

when K is known, as it was able to select the fewest features while outperforming k-means on the hardest synthetic datasets. This finding highlights a gap in the current clustering literature, which has focused primarily on centroid approaches.

The second half of this chapter introduced a more comprehensive two-/three-stage medoid approach, which addressed key limitations of the proposed medoid representation by: using an estimate of K, K_{est}, to guide the PSO search process; proposing a further improved multi-faceted fitness function that encourages good cluster quality, minimises m', and reduces the search space of K; and introducing a pseudo-local search, which refines the clusters produced by the second stage. These improvements were shown to significantly improve upon the K found by D-Medoid on datasets with high K. The two- and three-stage approaches also demonstrated a more appropriate trade-off between feature selection and clustering performance, selecting more features than D-Medoid but also achieving significantly better performance on the synthetic clustering datasets.

In future work, we will further refine our fitness function by taking a multi-objective approach in order to allow more intelligent balancing of all the three criteria: cluster performance, feature selection, and deduction of K. We would also like to investigate other methods of measuring cluster performance (perhaps using multi-objective techniques), such as connectedness or density. There is also scope for improving performance further with other methods for estimating K, penalising the number of features produced, or using other EC techniques or representations.

This chapter clearly demonstrated the ability of EC-based FS to improve clustering results. However, this is not the only way that EC can perform FM in clustering tasks; the next chapter will explore using EC-based FC for improving the performance and interpretability of clustering algorithms.

Chapter 4

Genetic Programming for Feature Construction in Clustering

4.1 Introduction

EC-based feature construction (FC) has been used to improve performance, reduce complexity, and improve model interpretability in a wide range of supervised learning tasks [37]. However, despite unsupervised learning posing even larger search spaces and interpretability concerns than supervised learning, there has been very few applications of EC-based FC in problems such as clustering [118]. Some existing GP work inadvertently perform embedded FC by using GP to perform clustering, but these methods do not leverage the full potential of GP to construct high-quality, sophisticated features. Indeed, no known EC work use the powerful wrapper FC approach, where features are automatically tailored to be specific to the clustering algorithm and dataset being used.

4.1.1 Chapter Goals

This chapter aims to propose the first GP approach to performing wrapper-based FC in clustering tasks. This approach is expected to improve clustering performance; reduce the number of features used by

clustering algorithms; and allow more interpretable clustering results utilising a small number of sophisticated, high-level constructed features. More specifically, this chapter will investigate:

- whether a basic wrapper-based GP method can be used to automatically produce multiple constructed features in a single run, and if so, which GP representation is most appropriate;
- if these constructed features are simple enough so that they can be understood;
- whether GP can be used to automatically generate tailored similarity functions, which are specific to a given clustering algorithm and dataset;
- if these evolved similarity functions improve performance while improving interpretability by using a subset of features; and
- whether evolving multiple heterogeneous similarity functions to make a consensus decision can further improve the results by allowing niching behaviour across the search space of a clustering problem.

4.1.2 Chapter Organisation

Sections 4.2 to 4.5 address the first two goals of this chapter by proposing two wrapper approaches using GP to automatically construct features for improving the performance of k-means clustering. The performance of these approaches are compared to k-means (using all features) across a number of datasets. Sections 4.6 to 4.10 tackle the third, fourth, and fifth goals, by developing a novel approach to automatically creating tailored similarity functions for use in clustering algorithms. Different varieties of this approach are evaluated in substantial depth across 17 clustering datasets compared to a wide range of existing clustering methods. A summary of the key findings of this chapter is provided in Section 4.11.

4.2 GP for Wrapper-Based FC in Clustering

k-means is the most well-known clustering algorithm due to its simple, efficient formulation and reasonable efficacy on basic problems [69]. However, it is fundamentally limited on more difficult problems where there are many dimensions/clusters or clusters are non-hyper-spherically shaped. By using GP to construct high-level features, the original feature space could be manipulated into a form which k-means can easily cope with. In this way, the performance of k-means can be improved beyond what is possible with the original features alone. Traditional GP program designs output only a single value from a single individual, meaning that only a single constructed feature (CF) is created. While a single CF may be adequate on easy datasets with a small K, when there are many clusters, it would be very difficult to accurately partition the dataset using a single value. Hence, new GP representations need to be developed to produce multiple CFs. The evolved system can also be taught to produce good clusters according to any measure of cluster quality, as GP individuals will learn to produce CFs to maximise the fitness of the wrapped k-means algorithm. In contrast, standard k-means simply minimises intra-cluster variance without considering any other indicators of cluster quality. As clustering will be performed on a constructed feature space, using a clustering algorithm more advanced than k-means may not be necessary, as GP should learn to produce features tailored to the clustering algorithm used.

In this section, we propose two new representations for performing FC using GP for clustering. We also introduce two fitness functions that can be used to train GP with k-means to improve the clustering performance. The overall design of the methods in this section is shown in Figure 4.1.

4.2.1 Multi-Tree Representation

To allow multiple CFs to be produced by a single GP individual, we propose an extension to the standard single-tree GP representation, so

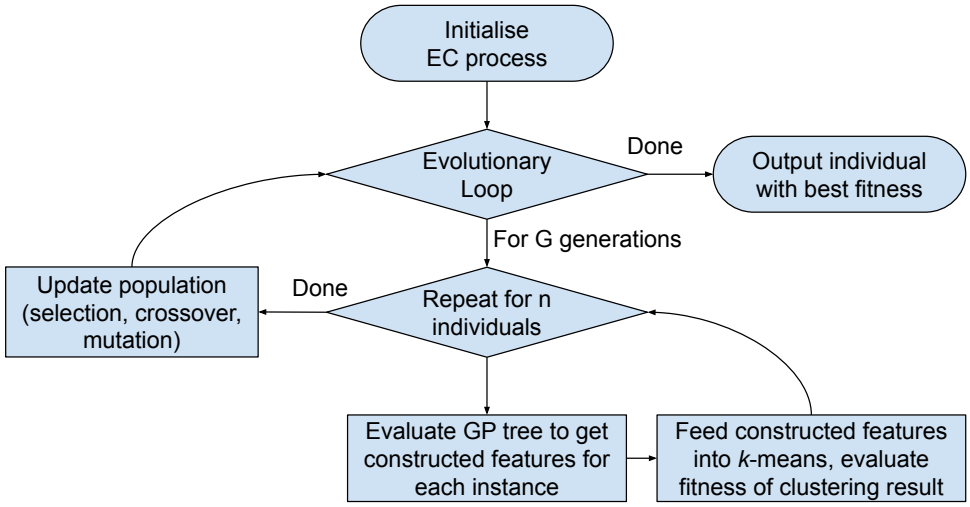


Figure 4.1: The overall flow of the GP wrapper-based FC methods. The multi-tree and vector representations differ in how they produce the constructed features for each instance.

that an individual contains multiple trees, producing multiple CFs. The number of trees (t) is dependent on the dataset used — generally, a higher K requires a higher t.

The function set used contains a number of standard arithmetic operators ($+, -, \times, \div, |+|, |-|$), as well as the max and min operators. Each of these operators take two children and produce a single output. The $\div$ operator is protected division: if the second child (the divisor) is 0, the operator returns 1. The final operator in the function set is if, which takes three children and returns the value of $child_2$ if $child_1$ is positive; otherwise it returns the value of $child_3$. The if operator is used to allow conditional behaviour in the GP program. The terminal nodes are the features of the dataset (f_1 through to f_m) as well as a random double in the range $[0, 1]$.

When such a multi-tree approach is used, the crossover and mutation operators in the evolutionary process must be adapted. In this work, we use a common approach [61, 162] whereby crossover is performed by selecting two individuals (in the normal way), selecting a random tree from each of the individuals, and then selecting a random sub-tree from each tree to use for crossover. Mutation is performed by choosing a

random tree from a random individual to be mutated.

While the multi-tree approach is reasonably straightforward to design, it has a number of limitations: most notably, that t must be set in advance. The crossover operator used may also be problematic: as any two random trees from a pair of individuals can be chosen for crossover, trees being crossed over may not correspond to similar CFs, and so the CFs produced are unlikely to be fully distinct from each other. Redundancy across a constructed feature set may affect the efficiency and interpretability of a given solution.

4.2.2 Vector Representation

To address the above issues, we also propose a single-tree approach which utilises a vector representation to produce multiple CFs from a single tree. The vector representation has no t parameter and so no parameter tuning is required. We use a similar function set as in the previous approach, but adapt each function to take two vectors as input and produce a vector as output. Each function operates on the input vectors in a pairwise manner, and the output vector has length equal to that of the smaller vector. We also introduce a $concat$ function that takes two vectors as input and outputs a vector that is the result of appending the second vector to the end of the first. This $concat$ function allows vectors of variable length to be generated, allowing GP to automatically generate a dynamic number of CFs. By using several $concat$ functions in a single tree, the constructed feature vector will grow as the tree is evaluated from bottom to top. The terminal set remains the same as in the previous approach, however each terminal node now outputs a **vector** containing the terminal value.

4.2.3 Fitness Function

When K is fixed, the most commonly used fitness function is the $\sum$ Intra fitness [3]:

$$\sum \text{Intra} = \sum_{i=1}^{K} \sum_{I_a \in C_i} d(I_a, Z_i) \tag{4.1}$$

where C_i represents the i^{th} cluster, $I_a \in C_i$ represents an instance in the i^{th} cluster, and Z_i represents the mean of the i^{th} cluster. This fitness function is what is minimised by k-means — when K is known, we should encourage all clusters to be as compact as possible, by minimising $\sum Intra$. One limitation of this measure is that clusters are encouraged towards hyper-sphericality; clusters will be unlikely to form non-spherical shapes that can occur on certain datasets.

One way of avoiding this problem is to use a fitness function based on connectedness. Connectedness measures the extent to which instances are in the same cluster as their immediate neighbours; close instances are similar and should fall in the same cluster. We propose a new fitness function, based on that proposed by Handl et al. [56], which computes the mean connectedness of all clusters in a partition:

$$\text{Connectedness} = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|C_i|} \sum_{I_a, I_b \in C_i} d_{inverse}(I_a, I_b) \tag{4.2}$$

$$d_{inverse}(I_a, I_b) = min\big[\frac{1}{d(I_a, I_b)}, 10\big] \tag{4.3}$$

The above fitness function (Equation (4.2)), which should be maximised, encourages clusters to contain instances which are close together. Equations (4.2) and (4.3) contain a number of extensions to the one proposed by Handl et al. [56]:

1. Closer neighbours are weighted more strongly, by directly using the distance between neighbours in the fitness calculation. The inverse distance is capped to a maximum of 10 (i.e. when dist $\leq$ 0.1) to prevent very similar/identical instances overly affecting fitness. The value of 10 was chosen empirically through testing on a range of datasets.

2. The mean connectedness is calculated across the set of clusters, instead of summing over all instance pairs. This discourages solutions with one very large cluster (with very good connectedness) and several very small clusters.

3. The mean connectedness within a cluster is used (instead of summing), to prevent very close instances from being over-represented in the fitness.

It is anticipated that by using connectedness, GP will produce features that allow for non-hyper-spherical clustering — although k-means itself will create hyper-spherical clusters in terms of the CFs, the CFs created by GP need not be linear transformations of the original features. The ability of our wrapper approach to train k-means based on different fitness measures allows k-means to be adapted to perform well on datasets that it would otherwise struggle on, especially when it is used with only a few CFs.

4.3 Experiment Design

Each combination of the two representations (multi-tree and vector) and two fitness functions ($\sum$ Intra and Connectedness: Equations (4.1) and (4.2)) were evaluated on a range of datasets using a variety of metrics. k-means was evaluated as a baseline, using all features. Each method is non-deterministic, and so is run 30 times using different seeds, and the mean result for each metric is computed.

Table 4.1 shows the evolutionary parameters used for all the GP methods across all the datasets. For the multi-tree approach, t is set to 7 — this was found empirically to be the required number of trees in order to give good performance across all datasets. k-means is also run for 100 iterations, or until convergence is reached (i.e. when cluster centres do not move between iterations). The initial cluster centres for k-means are randomly selected from the dataset. The seed of k-means is determined

Table 4.1: GP parameter settings.

Parameter	Value	Parameter	Value
Generations	100	Crossover Rate	80%
Population Size	1024	Mutation Rate	20%
Minimum Depth	2	Elitism	top-10
Maximum Depth	8	Selection Type	Tournament
Initial Population	Half-and-half	Tournament Size	7

Table 4.2: Datasets used in the experiments.

Real-World UCI datasets from [31].				Synthetic datasets from [56].			
Name	No. of Features	No. of Instances	No. of Classes	Name	No. of Features	No. of Instances	No. of Classes
Iris	4	150	3	10d10c	10	2730	10
Wine	13	178	3	10d20c	10	1014	20
Movement Libras	90	360	15	10d40c	10	1938	40
				50d10c	50	2699	10
Breast Cancer	9	683	2	50d20c	50	1255	20
				50d40c	50	2335	40
Image Segmentation	18	683	7	100d10c	100	2893	10
				100d20c	100	1339	20
Dermatology	34	359	6	100d40c	100	2212	40

using a hashing function applied to a GP tree so that each tree produces consistent partitions.

4.3.1 Datasets

A range of synthetic and real-world datasets were used to comprehensively evaluate the proposed methods, as shown in Table 4.2 Datasets were scaled so that each feature had values in $[0, 1]$, to prevent bias towards features with large ranges. These are the same datasets as used in Chapter 3; justification around the choice of these datasets is provided in Section 3.3 (page 73).

4.3.2 Evaluation Metrics

Clustering performance is measured using the two *internal* metrics defined previously, which directly measure the quality of a cluster partition. Connectedness (see Equation (4.2)) evaluates how well neighbouring instances are allocated to the same cluster, and $\sum$ Intra Distance (see Equation (4.1)) indicates how compact the clusters are.

In addition, we use two *external* metrics to measure how well the cluster partitions produced correspond to the dataset's class labels. These are class purity, which measures the homogeneity of each cluster with respect to the class labels, and the F-measure, which measures how well pairs of instances agree in terms of the clusters they are allocated to and their class labels. These measures are previously defined in Equations (2.15) and (3.9) respectively, on pages 35 and 76.

4.4 Results and Analysis

Tables 4.3 and 4.4 show the performance of the four GP methods and k-means (using all features (AF)) across the six real-world and nine synthetic datasets respectively. MTConn and MTIntra are the multi-tree approaches using the connectedness and $\sum$ Intra fitness function respectively, with $t = 7$. VectorConn and VectorIntra are the two vector approaches, each using one of the fitness functions proposed. Four metrics are shown: Conn (connectedness), $\sum$ Intra ($\sum$ intra distance), Purity (class purity), and FM (the F-measure). The $\sum$ Intra metric should be minimised (it is marked with a $*$ in the table); the other three metrics are maximised. For the GP methods, each result is labelled with a "$+$" or a "$-$" if it is significantly better or worse than the k-means baseline according to a Student's t-test performed with a 95% confidence interval. A lack of a "$+$" or "$-$" indicates no significant difference.

4.4.1 Results on Real-World Datasets

The GP methods generally perform well compared to k-means across the real-world datasets. All the four GP methods are significantly better in terms of the F-measure on the Iris, Wine, and Image Segmentation datasets. At least one of the GP methods is significantly better than k-means on all the remaining real-world datasets; GP is only significantly worse than k-means when using connectedness on the Breast Cancer dataset, where the $\sum$ Intra fitness measure gives much better performance. The connectedness fitness measure gives very good results on the Dermatology dataset, improving performance over k-means significantly. Clearly, different datasets require the use of different fitness functions: this flexbility is a key benefit of our proposed approach compared to the original k-means algorithm. Both the multi-tree and the vector approaches appear to perform similarly on the real-world datasets, with an exception on the Iris dataset, where the vector approach is superior when connectedness is used in terms of the external metrics.

Table 4.3: Performance on Real-World Datasets.

Method	Conn	Intra *	Purity	FM	Conn	Intra *	Purity	FM
	Iris				Wine			
MTConn7	223.4^{+}	29.59^{+}	0.8989^{+}	0.8308^{+}	90.13^{+}	88.99^{-}	0.9723^{+}	0.9444^{+}
MTIntra	26.49^{-}	29.28^{+}	0.8867^{+}	0.8111^{+}	7.621^{+}	88.7^{+}	0.9663^{+}	0.933^{+}
VectorConn	223.1^{+}	29.59^{+}	0.9502^{+}	0.9086^{+}	90.12^{+}	89.01^{-}	0.9697^{+}	0.9392^{+}
VectorIntra	26.49^{-}	29.28^{+}	0.8867^{+}	0.8111^{+}	7.618^{+}	88.7^{+}	0.9661^{+}	0.9325^{+}
k-means AF	26.77	31.39	0.8116	0.7544	7.561	88.74	0.9491	0.8998
	Movement Libras				Breast Cancer			
MTConn7	19.28^{+}	424.9^{-}	0.4424^{-}	0.3417	895.8^{+}	369.7^{-}	0.9101^{-}	0.8445^{-}
MTIntra	5.473^{+}	400.2^{+}	0.472	0.3527	15.63^{+}	331.6^{+}	0.9675^{+}	0.9423^{+}
VectorConn	19.1^{+}	414.3	0.4583	0.3434	898.1^{+}	376.7^{-}	0.8972^{-}	0.824^{-}
VectorIntra	5.486^{+}	399.0^{+}	0.4749^{+}	0.3542^{+}	15.64^{+}	331.6^{+}	0.9669^{+}	0.9413^{+}
k-means AF	5.134	414.5	0.4619	0.3439	15.52	332.0	0.9609	0.9313
	Image Segmentation				Dermatology			
MTConn7	798.4^{+}	877.1^{+}	0.6832^{+}	0.5886^{+}	42.28^{+}	377.1^{+}	0.946^{+}	0.9324^{+}
MTIntra	25.39^{+}	869.5^{+}	0.6654^{+}	0.5717^{+}	3.176^{+}	376.2^{+}	0.8655^{+}	0.7915
VectorConn	797.1^{+}	873.8^{+}	0.6859^{+}	0.5894^{+}	41.7^{+}	382.8	0.9063^{+}	0.8764^{+}
VectorIntra	25.38^{+}	872.2^{+}	0.6655^{+}	0.5726^{+}	3.063	380.9^{+}	0.8538	0.7839
k-means AF	24.78	908.6	0.6383	0.5582	3.022	386.5	0.8349	0.7569

4.4.2 Results on Synthetic Datasets

The GP methods continue to perform well compared to k-means on the synthetic datasets. All of the four methods have a significantly higher F-measure value than k-means on the datasets with 20 or 40 clusters. These datasets are the most difficult as they require separating the dataset into the greatest number of distinct groups. k-means performs very poorly when there is a large number of clusters (e.g. $K = 40$); GP is able to effectively perform FC to significantly improve the performance of k-means on the hardest datasets, while only using a small subset of the feature set. Some GP methods perform significantly worse on the simple 10d10c and 50d10c datasets, but at least one GP method is still significantly better than k-means in these cases.

The connectedness and $\sum$ Intra fitness measures are again superior on different datasets. The method using connectedness are significantly better than k-means on the 50d10c dataset, whereas those using $\sum$ Intra fitness are significantly worse. The inverse is true on the 10d10c dataset, however. In general, the multi-tree approach seems slightly better than the vector approach, especially on the datasets with highest dimensionality such as 100d20c and 100d40c. Future testing is required to evaluate which method is superior, and more work could be done to improve each method by further exploring alternative representations or fitness functions.

4.5 Evolved Program Analysis

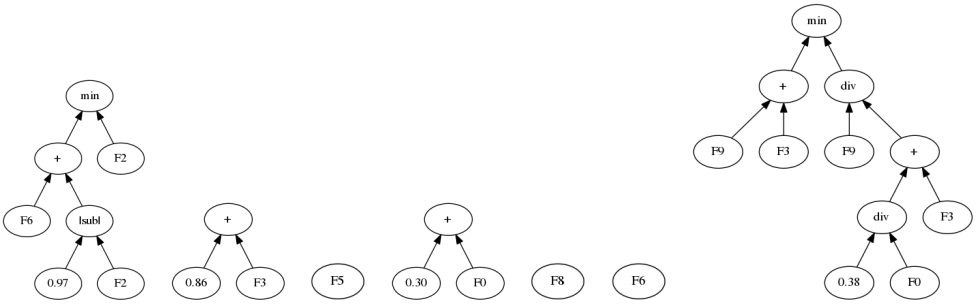
It is often useful when using GP to evaluate and analyse some of the high-performing individuals produced during the evolutionary process. Doing so allows us to understand what properties of a given tree allow it to perform well, which leads to a better understanding of the problem as well as allowing the GP method to be improved further. In addition, analysing evolved programs increases the confidence in our proposed method by demonstrating how it is able to achieve the good results we

Table 4.4: Performance on Synthetic Datasets.

Method	Conn	Intra *	Purity	FM	Conn	Intra *	Purity	FM
	10d10c				10d20c			
MTConn	823.3^{+}	719.1^{-}	0.9019^{-}	0.7836^{-}	177.0^{+}	213.2^{+}	0.9948^{+}	0.9919^{+}
MTIntra	17.67^{-}	710.1^{+}	0.9294	0.878	16.5^{+}	213.0^{+}	0.9948^{+}	0.9919^{+}
VectorConn	827.3^{+}	713.9	0.9153^{-}	0.8025^{-}	177.0^{+}	213.5^{+}	0.9941^{+}	0.9887^{+}
VectorIntra	18.05^{+}	706.3^{+}	0.9404^{+}	0.8926^{+}	16.48^{+}	213.5^{+}	0.9938^{+}	0.9906^{+}
k-means AF	17.88	712.4	0.9291	0.8571	15.29	254.8	0.8732	0.7969
	10d40c				50d10c			
MTConn	173.2^{+}	406.5^{+}	0.9747^{+}	0.9311^{+}	589.4^{+}	1480.0^{-}	0.7325^{-}	0.5167^{+}
MTIntra	16.34^{+}	405.0^{+}	0.977^{+}	0.9456^{+}	17.14^{-}	1220.0^{+}	0.7392	0.4785^{-}
VectorConn	173.6^{+}	403.3^{+}	0.9789^{+}	0.9409^{+}	587.3^{+}	1437.0^{-}	0.7278^{-}	0.5005
VectorIntra	16.32^{+}	404.1^{+}	0.9771^{+}	0.9494^{+}	17.22^{-}	1216.0^{+}	0.7397	0.4795^{-}
k-means AF	15.75	436.8	0.9182	0.8628	17.49	1317.0	0.744	0.4939
	50d20c				50d40c			
MTConn	163.3^{+}	583.2^{-}	0.7138^{+}	0.4996^{+}	163.3^{+}	894.7^{-}	0.685	0.4397^{+}
MTIntra	17.43	493.8^{+}	0.7456^{+}	0.4776^{+}	18.95^{-}	833.8^{+}	0.6952^{+}	0.4269^{+}
VectorConn	162.5^{+}	555.7	0.7212^{+}	0.4832^{+}	165.4^{+}	850.4^{+}	0.7082^{+}	0.4106^{+}
VectorIntra	17.52	487.2^{+}	0.7412^{+}	0.4351^{+}	19.3^{+}	797.1^{+}	0.7165^{+}	0.3759^{+}
k-means AF	17.33	546.5	0.6868	0.3823	19.16	865.2	0.6791	0.2618
	100d10c				100d20c			
MTConn	521.8^{+}	2123.0^{-}	0.7598	0.5311	126.0^{+}	885.4^{-}	0.7084	0.4657^{+}
MTIntra	15.81^{+}	1776.0^{+}	0.7835^{+}	0.5825^{+}	13.66^{+}	764.9^{+}	0.7481^{+}	0.4598^{+}
VectorConn	519.5^{+}	2077.0^{-}	0.7595	0.5446	125.5^{+}	850.5	0.7122	0.4451^{+}
VectorIntra	15.89^{+}	1771.0^{+}	0.7839^{+}	0.5854^{+}	13.74^{+}	749.6^{+}	0.7466^{+}	0.4331^{+}
k-means AF	15.14	1968.0	0.748	0.5255	13.31	844.2	0.7033	0.38
	100d40c							
MTConn	114.8^{+}	1234.0^{-}	0.6963	0.4629^{+}				
MTIntra	14.18^{-}	1118.0^{+}	0.7181^{+}	0.462^{+}				
VectorConn	116.0^{+}	1159.0	0.7142^{+}	0.4418^{+}				
VectorIntra	14.45	1061.0^{+}	0.7344^{+}	0.4028^{+}				
k-means AF	14.55	1184.0	0.6904	0.2675				

claim. In this section, we analyse a number of GP trees with high F-measure across a range of datasets.

An example of the multi-tree approach (for 10d20c) can be seen in Figure 4.2. The seven trees produced by an individual with a very high F-measure value of 0.9947 are shown, along with the constructed feature set generated, which consists of seven features, one from each tree. Of these trees, three are simply performing feature selection of a single



(a) (b) (c) (d) (e) (f) (g)

The evolved multi-tree GP individual with seven trees.

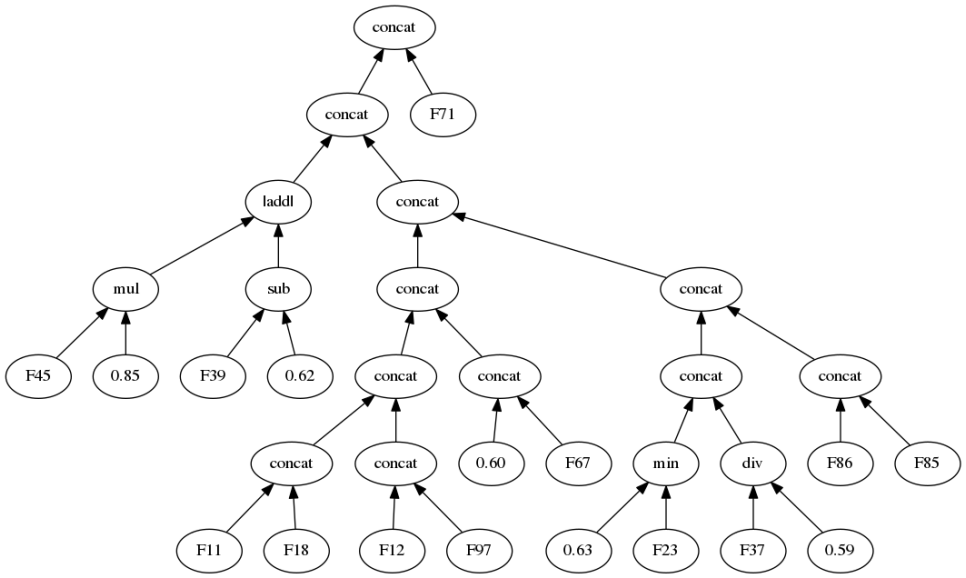
$$\Big[min(F_6 + |0.97 - F_2|, F_2), \quad (0.86 + F_3), \quad F_5, \quad (0.3 + F_0), \quad F_8, \quad F_6, \quad min\Big((F_9 + F_3), \frac{F9}{(\frac{0.38}{F_0} + F_3)}\Big)\Big]$$

(h) Constructed feature set generated by the programs.

Figure 4.2: An evolved *multi-tree* individual on the 10d20c dataset (FM: 0.9947).

feature, two add a constant value to a single feature, and the remaining two are performing more advanced FC. In total, seven of the original 10 features are used. Although the dimensionality has not been greatly reduced, performance is still much higher than that of the original k-means algorithm (which achieves an F-measure value of 0.7969). This further highlights the ability of GP to improve performance by selecting the most important features, and creating more powerful high-level features.

Figure 4.3 shows a GP individual using the vector approach with high performance on the hardest synthetic dataset (100d40c). The individual is a reasonably concise tree, with a maximum width of eight nodes and a depth of seven. The tree itself is shown in Figure 4.3a, and the output of the tree as shown in Figure 4.3b. The tree selects feature values as terminal nodes, and outputs a constructed feature vector of length 12, containing 11 "constructed" features and one constant value. Of these CFs, one is an arithmetic combination of two selected features and two constants, two are operations applied to a selected feature and a constant value, and the remaining nine are unchanged selected features. k-means



(a) The GP tree.

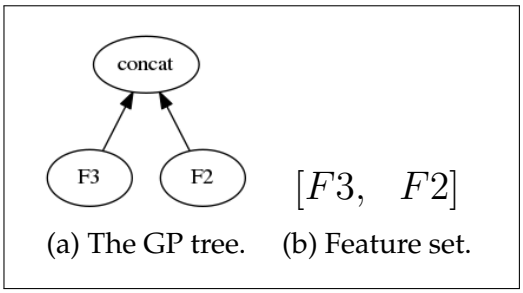
$\left[|(F_{45} \times 0.85) + (F_{39} - 0.62)|,\ F_{11},\ F_{18},\ F_{12},\ F_{97},\ 0.6,\ F_{67},\ min(0.63, F_{23}),\ \frac{F_{37}}{0.59},\ F_{86},\ F_{85},\ F_{71}\right]$

(b) Constructed feature set generated by the program.

Figure 4.3: An evolved *vector* individual on the 100d40c dataset (FM: 0.499).

achieved an F-measure value of 0.2618 on average; this GP individual produced nearly double the F-measure value (0.499) while only using 12 features compared to the 100 original features that k-means used. This large increase in performance shows the power of GP in improving performance by creating more powerful high-level features while also reducing dimensionality.

A useful property of the vector approach is its ability to dynamically produce a variable number of CFs. For example, on the Iris dataset, which has only three classes, it is unnecessary to have seven CFs (as occurs for $t = 7$ in the multi-tree approach) and having so many features may reduce the interpretability of the solution. Figure 4.4 shows a high performing, very simple GP individual produced on the Iris dataset. This tree is very easy to analyse: it simply selects two of the four features in the dataset (F_3 and F_2). By not selecting the other misleading or



(a) The GP tree. (b) Feature set.

Figure 4.4: An evolved *vector* individual on the Iris dataset (FM: 0.9233).

redundant features, this GP tree significantly improves the ability of k-means to produce a good cluster partition.

4.6 Evolving Similarity Functions for Clustering using GP

The wrapper-based approaches proposed in Section 4.2 showcased the potential of GP to improve the performance and interpretability of clustering algorithms through the automatic construction of high-level features. However, the clustering algorithm (k-means) was forced to weight all constructed features equally in all cases, as the similarity function was pre-determined to be Euclidean distance. It is not likely that this is the optimal similarity function to use, especially across a range of different datasets with varying characteristics. Ideally, we would hope that the best similarity function for a given clustering algorithm and dataset could be automatically found, but this in itself is clearly an NP-hard task if we allow arbitrary similarity functions.

The clustering literature has an overwhelming focus on producing novel clustering algorithms, which employ a wide range of techniques for modelling and searching the clustering problem space. However, there has been very little focus on new techniques for automatically creating more appropriate and more powerful similarity measures to accurately model the relationships between instances on a specific dataset. GP, with its intrinsic function-like solution structure, is a natural candidate for

automatically evolving similarity functions tailored to the data it is trained on. GP, and EC methods in general, have been shown to be effective on large dataset sizes and dimensionality; GP has the potential to evolve smaller, more refined, and more interpretable similarity functions on very big datasets. The remainder of this chapter investigates the capability of GP for automatically constructing power similarity functions.

4.7 Evolving Similarity Functions: Proposed Approaches

A variety of representations have been proposed for modelling clustering solutions. The graph representation models the data in an intuitive way, where instances (nodes) are connected by an edge if they are *similar enough* [174]. This is a powerful representation that allows modelling a variety of cluster shapes, sizes, and densities, unlike the more common prototype-based representations such as k-means. However, algorithms using graph representations are very dependent on the criterion used to select edges [174]. One of the most common criteria is to simply use a fixed threshold [174], which indicates the distance at which two instances are considered too dissimilar to share an edge. Such a threshold must be determined independently for every dataset, and this approach typically does not allow varying thresholds to be used in different clusters. Another popular criterion is to put an edge between each instance and its N-nearest neighbours [174], where N is a small integer value such as 2, 3, or 4. N must also be determined before running the algorithm, with results being very sensitive to the N value chosen.

Given that graph-based clustering performance is very dependent on a good choice of similarity function (and parameter tuning), we focus on this type of algorithm as a good candidate for improving through the use of automatically evolved similarity functions. An overview of the proposed GP for Graph-based Clustering (GPGC) algorithm is shown in

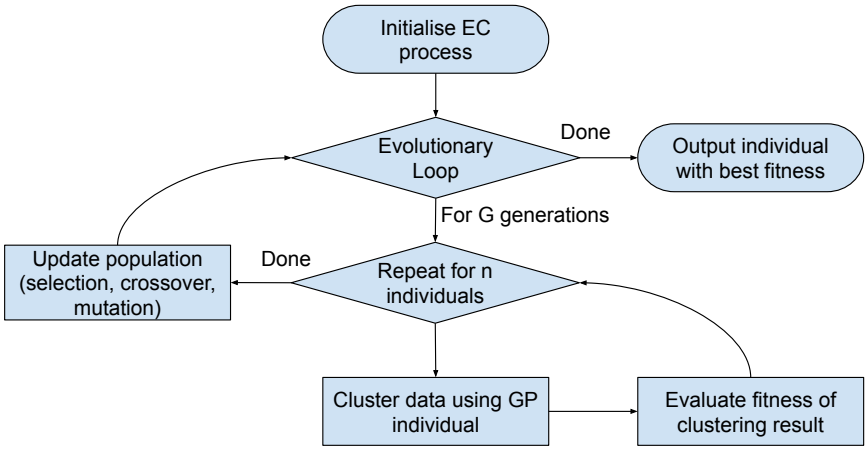


Figure 4.5: The overall flow of the proposed GPGC algorithm. The clustering process is discussed in detail in Section 4.7.2, and is shown in Algorithm 1.

Figure 4.5. We discuss the different parts of this overall algorithm in the following subsections.

4.7.1 GP Representation

To represent a similarity function, a GP tree must take two instances as input and produce a single floating-point output corresponding to the similarity of the two instances. Therefore, we define the terminal set as all feature values of both instances, such that there are $2m$ possible terminals for m features (I_0F_0 and I_1F_0 through to I_0F_{m-1} and I_1F_{m-1}), as well as a random floating-point value (for scaling purposes). The function set comprises of the normal arithmetic functions ($+$, $-$,$\times$,protected $\div$), two absolute arithmetic functions ($|+|$ and $|-|$), and the max, min and if operators. All of these functions asides from if take two inputs and output a single value, which is the result of applying that function. The if function takes three inputs and outputs the second input if the first is positive, or the third input if it is not. We include the if, max, and min functions to allow conditional behaviour within a program, in order to allow the creation of similarity functions that operate differently across the feature space. The $\div$ operator is protected

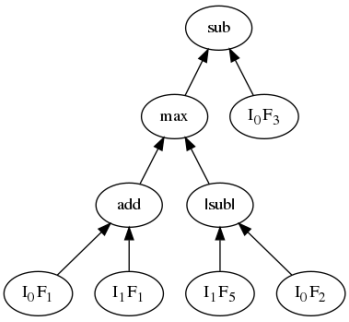


Figure 4.6: An example of of a similarity function with the expression $sub\big(max(add(I_0F_1, I_1F_1), |sub|(I_1F_5, I_0F_2)), I_0F_3\big)$.

division: if the divisor (the second input) is zero, the operator will return a value of one. An example of a similarity function using this GP program design is shown in Figure 4.6.

4.7.2 Clustering Process

As we are using a graph representation, every pair of instances that are deemed "close enough" by an evolved GP tree should be connected by an edge. As discussed before, we would like to refrain from using a fixed similarity threshold as varying thresholds may be required across a dataset due to varying cluster density. We therefore use the approach where each instance is connected to a number of its most similar neighbours (according to the evolved similarity function).

To find the most similar neighbour of a given instance for an evolved similarity function requires comparing the instance to every other instance in the dataset. Normally, when using a distance metric, these pairwise similarities could be precomputed; however, in the proposed algorithm, these must be computed separately for every evolved similarity function, giving $O(n^2)$ comparisons for every GP individual on every generation of the training process, given n instances. In order to reduce the computational cost, we use a heuristic whereby each instance is only compared to its l nearest neighbours based on Euclidean distance. The set of nearest neighbours can be computed at the start of the EC training process, meaning only $O(nl)$ comparisons are required per GP

individual. By using this approach, we balance the flexibility of allowing an instance to be connected to many different neighbours with the efficiency of using a subset of neighbours to compare to. As we use Euclidean distance only to give us the *order* of neighbours, the problems associated with Euclidean distance at high dimensions should not occur. We found in practice that setting l as $l = \lceil \sqrt[3]{n} \rceil$ gave a good neighbourhood size that is proportional to n, while ensuring l is at least 2.

Algorithm 1 shows the steps used to produce a cluster for a given GP individual, X. For each instance I in the dataset, the nearest l neighbours are found using the pre-computed Euclidean distance mappings. Each of these l neighbours is then fed into the bottom of the tree (X) along with I. The tree is evaluated, and produces an output corresponding to the similarity between I and that neighbour. The neighbour with the highest similarity is chosen, and an edge is added between it and I. As in GPGC, we tested adding edges to more than one nearest neighbour, but found that performance decreased. Once this process has been completed for each $I \in Dataset$, the set of edges formed will give a set of graphs, where each graph represents a single cluster. These graphs are converted to a set of clusters by assigning all instances in each graph to the same cluster.

4.7.3 Fitness Function

The most common measures of cluster quality are compactness and separability [3]. A good cluster partition should have distinct clusters that are very dense in terms of the instances they contain, and that are far away from other clusters. A third, somewhat less common measure, is the instance connectedness, which measures how well a given instance lies in the same cluster as its nearby neighbours [56]. The majority of the clustering literature measures performance in a way that implicitly encourages hyper-spherical clusters to be produced, by minimising each instance's distance to its cluster mean, and maximising the distance between different cluster means. Such an approach is problematic, as it introduces bias in the shape of clusters produced, meaning elliptical or

Algorithm 1: Process to produce a cluster using a given GP individual (X) and the number of neighbours (l).

```
Edges = {};
for I ∈ Dataset do Choose edge
    Neighbours = nearestNeighbours(I, l);
    Neighbour_Best = ∅;
    Similarity_Best = −∞;
    for Y ∈ Neighbours do Test neighbour
        similarity = evaluate(X, I, Y);
        if similarity > Similarity_Best then
            Similarity_Best = similarity;
            Neighbour_Best = Y;
        end
    end
    add edge from I to Neighbour_Best to Edges;
end
Cluster = graphToCluster(Edges);
```

other non-spherical clusters are unlikely to be found.

As a graph representation is capable of modelling a variety of cluster shapes[1], we instead propose using a fitness function that balances these three measures of cluster quality in a way that gives minimal bias to the shape of clusters produced. We discuss each of these in turn below.

Compactness To measure the *compactness* of a cluster, we choose the instance in the cluster that is the furthest away from its nearest neighbour in the same cluster; that is, the instance that is the most isolated within the cluster. The distance between that instance and its nearest neighbour, called the *sparsity* of the cluster, should be minimised. We define sparsity in Equation (4.4), where C_i represents the i^{th} cluster of K clusters, $I_a \in C_i$ represents an instance in the i^{th} cluster, and $d(I_a, I_b)$ is the Euclidean distance between two instances.

$$\text{Sparsity} = \max_{I_a \in C_i} \left\{ \min_{I_b \in C_i} d(I_a, I_b) \middle| I_a \neq I_b \right\} \tag{4.4}$$

[1]Examples of a range of cluster shapes can be seen in Figure 2.1 on page 24.

Separability To measure the separation of a cluster, we find the minimum distance from that cluster to any other cluster. This is equivalent to finding the minimum distance between the instances in the cluster and all other instances in the dataset that are not in the same cluster, as shown in Equation (4.5). The separation of a cluster should be maximised to ensure that it is distinct from other clusters.

$$\text{Separation} = \min_{I_a \in C_i} \Big\{ \min_{I_b \notin C_i} d(I_a, I_b) \Big\} \tag{4.5}$$

Connectedness An instance's *connectedness* is measured by finding how many of its c nearest neighbours are assigned to the same cluster as it, with higher weighting given to neighbours that are closer to the given instance, as shown in Equation (4.6). To prevent connectedness from encouraging spherical clusters, c must be chosen to be adequately small — otherwise, large cluster "blobs" will form. We found that setting $c = 10$ provided a good balance between producing connected instances and allowing varying cluster shapes. The mean connectedness of a dataset should be maximised.

$$\text{Connectedness} = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|C_i|} \sum_{I_a \in C_i,\ I_b \in N_{I_a}} d_{inv}(I_a, I_b) \big| I_b \cap C_i \tag{4.6}$$

where N_{I_a} gives the c nearest neighbours of I_a, $I_b \cap C_i$ indicates that I_a and I_b are (correctly) in the same cluster, and

$$d_{inv}(I_a, I_b) = min\big[\frac{1}{d(I_a, I_b)}, 10\big] \tag{4.7}$$

The inverse distance between two instances is capped at 10, to prevent very close instances from overly affecting the fitness measure. Inverse distance is used to weight closer neighbours more highly.

Our proposed fitness function is a combination of these three measures (Equations (4.4) to (4.6)): we find each cluster's ratio of sparsity: separation (as they are competing objectives) as shown in Equation (4.8), and then measure the partition's fitness by also considering the

connectedness, as shown in Equation (4.9). This fitness function should be maximised.

$$\text{Mean SpaSep} = \frac{1}{K}\sum_{i=1}^{K}\frac{\text{Sparsity}}{\text{Separation}} \tag{4.8}$$

$$\text{Fitness} = \frac{\text{Connectedness}}{\text{Mean SpaSep}} \tag{4.9}$$

4.7.4 Using a Multi-Tree Approach

GP can also use multiple trees to represent each individual/solution. Multi-tree GP has the potential to automatically generate multiple complementary similarity functions, which are able to *specialise* on different clusters in the dataset.

As previously discussed, using a single fixed similarity function means that every pair of instances across a dataset must be compared identically, i.e. with all features weighted equally regardless of the characteristics of the given instances. By using GP to automatically evolve similarity functions containing conditional nodes (if, max, and min), we are able to produce trees that will measure similarity dynamically. However, a tree is still limited in its flexibility, as there is an inherent trade-off between the number of conditional nodes used and the complexness of the constructed features in a tree — more conditionals will tend to mean simpler constructed features with fewer operators (and vice versa), due to the limitations on tree depth and training time.

To tackle these issues, while still maintaining reasonable tree depth and training time, we propose evolving a number of similarity functions concurrently. Using this approach, a pair of instances will be assigned a similarity score by each similarity function, which are then summed together to give a total measure of how similar the two instances are[2]. In

[2]Of course, there are a variety of ways to combine similarity scores in the ensemble learning literature, such as taking the maximum output. We use the sum here to increase the stability of the joint similarity function, and to reduce the effect of outliers/edge

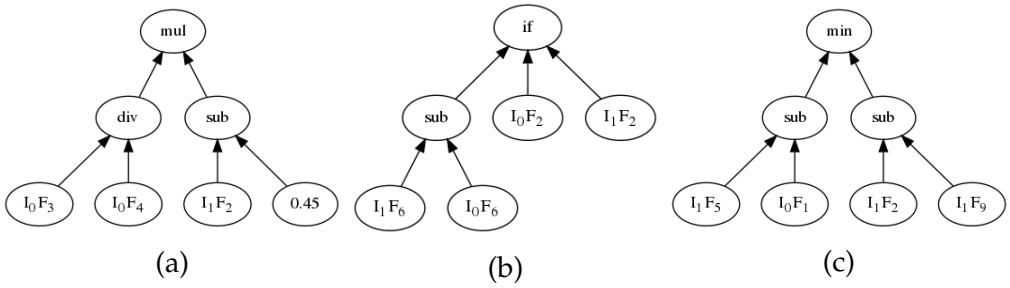


Figure 4.7: An example of a multi-tree similarity function.

this regard, each similarity function provides a measure of its *confidence* that two instances should lie in the same cluster, allowing different similarity functions to specialise on different parts of the dataset. This is implemented using GP with a multi-tree approach, where each GP individual contains not only one but multiple trees. An example of this structure is shown in Figure 4.7 with the number of trees, $t = 3$. As all t similarity functions are evolved concurrently in a single individual, a set of cohesive functions will be evolved that work well together, but that are not expected to be good similarity functions independently. In this way, a GP individual can be thought of as a *meta-function*. The core of the clustering process remains the same with this approach, with the only change being that the most similar neighbour for a given instance is based on the sum of similarities given by all trees in an individual. This change to the algorithm is shown in Algorithm 2.

There are several factors that must be considered when extending the proposed algorithm to use a multi-tree approach: how to perform crossover when there are multiple trees to crossover between, and how many trees to use. These two factors will be discussed in the following paragraphs. A third consideration is the maximum tree depth — we use a smaller tree depth when multiple trees are used, as each tree is able to be more specialised and so does not require as many nodes to produce a good similarity function. Mutation is performed as normal, by randomly choosing a tree to mutate.

cases. We hope to investigate this further in future work.

Algorithm 2: Choosing the most similar neighbour to an instance (I) in the multi-tree approach for individual X.

```
for Y ∈ Neighbours do Test neighbour
    similarity_sum = 0;
    for T ∈ X do Each tree
        similarity_sum += evaluate(T, I, Y);
    end
    if similarity_sum > Similarity_Best then
        Similarity_Best = similarity_sum;
        Neighbour_Best = Y;
    end
end
```

Crossover Strategy

In standard GP, crossover is performed by selecting two individuals, randomly selecting a subtree from each of these two individuals, and swapping the selected subtrees to produce new offspring. In multi-tree GP, a tree within each individual must also be selected. There are a number of possible methods for doing so [61, 162], as discussed below:

Random-index crossover: The most obvious method is to randomly select a tree from each individual (*random-index crossover*) (RIC). This method may be problematic when applied to our proposed approach, as it reduces the ability of each tree to specialise, by exchanging information between trees that may have different "niches".

Same-index crossover: An alternative method to avoid the limitations of RIC is to always pick two trees at the same index in each individual. For example, selecting the third tree in both individuals. This method, which we call *same-index crossover* (SIC), allows an individual to better develop a number of distinct trees while still encouraging co-operation between individuals through the crossover of related trees.

All-index crossover: The SIC method can be further extended by performing crossover between every pair of trees simultaneously, i.e.

crossover between every i^{th} tree in both individuals, where $i \in [1, t]$ for t trees. This *all-index crossover* (AIC) approach performs information exchange more aggressively between individuals, which should increase training efficiency. However, it introduces the requirement that the effect of performing all pairs of crossovers gives a net fitness increase, which may limit the exploitation of individual solutions during the EC process.

We will compare these crossover approaches to investigate which type of crossover is most appropriate.

Number of Trees

The number of trees used in a multi-tree approach must strike a balance between the performance benefit gained by using a large number of specialised trees and the difficulty in training many trees successfully. When using either the SIC or RIC crossover methods, increasing the number of trees used will reduce the chance proportionally that a given tree is chosen for crossover/mutation, thereby decreasing the rate at which each tree is refined. When the AIC method is used, a larger number of trees increases the probability that a crossover will not improve fitness, as the majority of the trees are unlikely to gain a performance boost when crossed over in the later stages of the training process when small "tweaks" to trees are required to optimise performance. We will investigate the effect of the number of trees used on the fitness obtained later in this chapter.

4.8 Evolving Similarity Functions: Experiment Design

4.8.1 Benchmark Techniques

We compare our proposed single-tree approach (GPGC) to a variety of baseline clustering methods, which are listed below. We also compare the single- and multi-tree approaches to investigate the effectiveness of using

additional trees.

- k-means++ [12], a commonly used partitional clustering algorithm. Standard k-means++ cannot automatically discover the number of clusters, and so K is pre-fixed for this method. We use this as an example of a relatively simple and widely used method in the clustering literature.

- OPTICS [11], a well-known density-based algorithm. OPTICS requires a contrast parameter, ξ, to be set in order to determine where in the dendrogram the cluster partition is extracted from; we test OPTICS with a range of ξ values in $[0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]$ and report the best result in terms of the Adjusted Rand Index (defined in Equation (2.13)).

- Two naïve graph-based approaches that connect every instance with an edge to its n-nearest neighbours [174]. We test with both $n = 2$ (called NG-2NN) and $n = 3$ (called NG-3NN) in this work. Note that the case where $n = 1$ (NG-1NN) is similar to the clustering process used in Section 4.7.2; we exclude NG-1NN as it produces naïve solutions with a fixed distance function.

- The Markov Clustering (MCL) algorithm [172] is another clustering algorithm using a graph-based representation, which simulates random walks through the graph and keeps instances in the same cluster when they have a high number of paths between them.

- The multi-objective clustering with automatic k-determination (MOCK) algorithm [56], as an example of a well-known high-quality EC clustering method.

4.8.2 Datasets

We use a range of synthetic clustering datasets to evaluate the performance of our proposed GPGC approach, with varying cluster

Table 4.5: Datasets generated using a Gaussian distribution [56].

Name	m	n	K
10d10cGaussian	10	2730	10
10d20cGaussian	10	1014	20
10d40cGaussian	10	1938	40

Table 4.6: Datasets generated using an Elliptical distribution [56].

Name	m	n	K	Name	m	n	K
10d10c	10	2903	10	100d10c	100	2893	10
10d20c	10	1030	20	100d20c	100	1339	20
10d40c	10	2023	40	100d40c	100	2212	40
10d100c	10	5541	100	1000d10c	1000	2753	10
50d10c	50	2699	10	1000d20c	1000	1088	20
50d20c	50	1255	20	1000d40c	1000	2349	40
50d40c	50	2335	40	1000d100c	1000	6165	100

shapes, numbers of features (m), instances (n) and clusters (K). We avoid using real-world classification datasets as done in previous clustering studies, as there is no requirement that classes should correspond well to homogeneous clusters [175] — for example, clustering the well-known Iris dataset will often produce two clusters, as the versicolor and virginica classes overlap significantly in the feature space. The datasets were generated with the popular generators introduced by Handl et al. [56]. The first generator uses a Gaussian distribution, which produces a range of clusters of varying shapes at low dimensions, but produces increasingly hyper-spherical clusters as m increases. As such, we use this generator only at a small m, to produce the datasets shown in Table 4.5.

The second generator produces clusters using an elliptical distribution, which produces non-hyper-spherical clusters even at large dimensionality. A wide variety of datasets were generated with this distribution, with m varying from 10 to 1000, and K varying from 10 to

Table 4.7: Common GP Parameter Settings.

Parameter	Setting	Parameter	Setting
Generations	100	Population Size	1024
Mutation	20%	Crossover	80%
Elitism	top-10	Selection Type	Tournament
Min. Tree Depth	2	Max. Tree Depth	5 (MT), 7 (ST)
Tournament Size	7	Pop. Initialisation	Half-and-half

100, as shown in Table 4.6. Datasets with $K = 10$ clusters have between 50 and 500 instances per cluster, whereas datasets with a higher K have between 10 and 100 to limit the memory required. These datasets allow our proposed approach to be tested on high-dimensional problems. All datasets are scaled so that each feature is in $[0, 1]$ to prevent feature range overly affecting the distance calculations used in the clustering process. As a generator is used, the cluster that each instance is assigned to is known — i.e. the datasets provide a *gold standard* in the form of a "cluster label" for each instance. While this label is not used during training, it is useful for evaluating the clusters produced by the clustering methods.

4.8.3 Parameter Settings

The non-deterministic methods (k-means++, GPGC, MOCK, MCL) were run 30 times, and the mean results were computed. k-means++, GPGC and MOCK were run for 100 iterations, by which time k-means++ had achieved convergence. All benchmarks use Euclidean distance. The GP parameter settings for the single- and multi-tree GPGC methods, are based on standard parameters [135], and are shown in Table 4.7; the multi-tree (MT) approach uses a smaller maximum tree depth than the single-tree (ST) approach due to having multiple, more-specific trees. The MOCK experiments used the attainment score method to select the best solution from the multi-objective approximation front.

4.8.4 Evaluation Metrics

To evaluate the performance of each of the clustering algorithms, we use the three measures defined previously (connectedness, sparsity, and separation), as well as the Adjusted Rand Index (ARI), which compares the cluster partition produced by an algorithm to the gold standard provided by the cluster generators in an adjusted-for-chance manner. The ARI is defined in Equation (2.13) on page 34.

4.9 Evolving Similarity Functions: Results and Discussion

We provide and analyse the results of our experiments in this section. We begin by comparing each of the proposed multi-tree approaches to the single-tree GPGC approach in order to decide which version of GPGC is the more effective (Section 4.9.1). We then compare the best of these approaches, GPGC-AIC, to the benchmark methods to examine how well our proposed method performs relative to existing clustering methods (Section 4.9.2). The effect of the number of trees on the performance of the multi-tree approach is analysed in Section 4.9.4.

4.9.1 GPGC using Multiple Trees

To further improve the performance of the proposed GPGC approach, we proposed an extension to use a multi-tree GP design in Section 4.7.4. To analyse the effectiveness of this extension, and determine which type of multi-tree crossover is most effective, we evaluated the three crossover methods (RIC, SIC, AIC) against the single-tree GPGC approach. We used $t = 7$ trees based on initial tests — the effect of varying t is discussed further in Section 4.9.4. Tables 4.8 and 4.9 show the results of these experiments on the datasets generated using a Gaussian and elliptical distribution respectively. For each of the four methods, we provide the (mean) number of clusters (K), as well as four metrics of

Table 4.8: Crossover: Datasets using a Gaussian Distribution.

Dataset	Method	Fitness	K	Conn.	Spar.*	Sep.	ARI
10d10cGaussian	GPGC	19.23	21.5	41.9	0.293	0.140	0.750
	AIC	23.75^{+}	8.8	51.4^{+}	0.324^{-}	0.154^{+}	0.880^{+}
	RIC	24.47^{+}	8.1	52.4^{+}	0.324^{-}	0.156^{+}	0.859^{+}
	SIC	24.66^{+}	7.6	52.9^{+}	0.326^{-}	0.157^{+}	0.833^{+}
10d20cGaussian	GPGC	63.00	19.7	47.3	0.268	0.375	0.991
	AIC	63.79	19.5	47.3	0.268	0.377	0.980^{-}
	RIC	63.26	19.7	47.3	0.269	0.376	0.988
	SIC	63.33	19.7	47.3	0.268	0.376	0.991
10d40cGaussian	GPGC	57.81	34.8	48.6	0.267	0.331	0.958
	AIC	60.37^{+}	33.7	48.9^{+}	0.265^{-}	0.337^{+}	0.943^{-}
	RIC	60.05^{+}	34.0	48.9^{+}	0.266^{-}	0.336^{+}	0.955
	SIC	58.89	34.6	48.8	0.267	0.334	0.958

cluster quality: fitness achieved, connectedness (Conn), sparsity (Spar)[3], separation (Sep), and the ARI. Connectedness, sparsity, and separation are defined in the same way as in the fitness function. We performed a two-tailed Mann Whitney U-Test at a 95% confidence interval comparing each of the multi-tree approaches to the single-tree approach on each of the metrics. A "$+$" indicates a method is significantly better than the single-tree GPGC method, a "$-$" indicates it is significantly worse, and no symbol indicates no significant difference was found. For all metrics except for sparsity, a larger value indicates a better result.

The most noticeable result of using a multi-tree approach is that the fitness achieved by the GP process is significantly improved across all datasets with the exception of the 10d20cGaussian, 10d40c and 10d100c datasets, where the multi-tree approaches were significantly worse or had similar fitness to GPGC. On the datasets generated using a Gaussian distribution, the multi-tree approaches are able to find the number of clusters much accurately on 10d10cGaussian, and achieve a significantly higher ARI result. On the 10d40cGaussian dataset, both AIC and RIC achieved significantly better fitness, connectedness, sparsity, and separation than GPGC. While AIC is significantly worse than GPGC on

[3]Sparsity is labelled with a $*$ in tables as a reminder that it should be minimised.

Table 4.9: Crossover: Datasets using an Elliptical Distribution.

Dataset	Method	Fitness	K	Conn.	Spar.*	Sep.	ARI
10d10c	GPGC	19.17	36.4	61.2	0.157	0.061	0.737
	AIC	21.35^{+}	24.0	71.7^{+}	0.166^{-}	0.059	0.814^{+}
	RIC	20.68^{+}	23.1	73.7^{+}	0.168^{-}	0.058^{-}	0.799^{+}
	SIC	21.13^{+}	24.7	72.9^{+}	0.164^{-}	0.057	0.806^{+}
10d20c	GPGC	43.69	27.8	73.3	0.150	0.098	0.663
	AIC	49.11^{+}	22.5	77.4^{+}	0.154	0.106^{+}	0.666
	RIC	49.51^{+}	21.9	77.7^{+}	0.154	0.106^{+}	0.656
	SIC	50.28^{+}	21.8	78.3^{+}	0.155^{-}	0.106^{+}	0.677
10d40c	GPGC	36.57	55.6	70.9	0.136	0.079	0.579
	AIC	33.43^{-}	49.2	76.6^{+}	0.139	0.072^{-}	0.522
	RIC	31.92^{-}	53.2	75.1^{+}	0.138	0.070^{-}	0.486^{-}
	SIC	31.95^{-}	51.1	75.9^{+}	0.140	0.071^{-}	0.539
10d100c	GPGC	31.80	109.5	74.0	0.131	0.066	0.424
	AIC	32.40	106.4	75.6	0.131	0.066	0.421
	RIC	30.84	134.9	72.6	0.127	0.064	0.442
	SIC	31.78	113.4	74.1	0.130	0.066	0.444
50d10c	GPGC	31.66	12.6	57.8	0.445	0.276	0.962
	AIC	42.49^{+}	10.0	59.5^{+}	0.472^{-}	0.341^{+}	0.987^{+}
	RIC	41.21^{+}	10.0	60.1^{+}	0.465^{-}	0.328^{+}	0.977^{+}
	SIC	40.21^{+}	10.4	59.9^{+}	0.459	0.320^{+}	0.969
50d20c	GPGC	30.36	25.2	50.6	0.350	0.234	0.807
	AIC	36.78^{+}	21.1	51.8^{+}	0.359	0.273^{+}	0.837
	RIC	34.82^{+}	21.5	51.3	0.362^{-}	0.268^{+}	0.841
	SIC	34.06^{+}	22.1	51.3	0.358^{-}	0.261^{+}	0.848
50d40c	GPGC	29.58	49.8	54.6	0.304	0.196	0.726
	AIC	34.36^{+}	45.2	56.1^{+}	0.308	0.220^{+}	0.810^{+}
	RIC	32.13^{+}	46.2	55.8^{+}	0.306	0.209^{+}	0.721
	SIC	32.27^{+}	46.3	55.8^{+}	0.307	0.208^{+}	0.776

10d20cGaussian and 10d40cGaussian, the decrease of ~1.5% ARI is not very meaningful given it gained 13% ARI on 10d10cGaussian.

The multi-tree approaches also tend to produce clusters that are both better connected and better separated than GPGC on the datasets generated with an elliptical distribution. It seems that using multiple trees allows the GP evolutionary process to better separate clusters, while still ensuring that similar instances are placed in the same cluster.

Table 4.9: Crossover: Datasets using an Elliptical Distribution (Part 2).

Dataset	Method	Fitness	K	Conn.	Spar.*	Sep.	ARI
100d10c	GPGC	39.40	10.4	47.9	0.611	0.545	0.993
	AIC	44.40^{+}	9.8	48.1	0.621	0.580^{+}	0.998
	RIC	44.88^{+}	9.7	48.2	0.622	0.584^{+}	0.997
	SIC	42.87	10.0	48.1	0.617	0.569	0.996
100d20c	GPGC	28.18	22.2	38.4	0.527	0.449	0.883
	AIC	32.00^{+}	20.6	38.2	0.535	0.494^{+}	0.917^{+}
	RIC	31.31^{+}	20.7	38.6	0.530	0.481^{+}	0.905
	SIC	31.59^{+}	20.5	38.3	0.538	0.492^{+}	0.921^{+}
100d40c	GPGC	21.60	50.2	39.7	0.436	0.282	0.724
	AIC	25.02^{+}	45.9	40.6^{+}	0.440	0.316^{+}	0.771
	RIC	24.50^{+}	47.6	40.7^{+}	0.438	0.311^{+}	0.777^{+}
	SIC	23.64^{+}	49.2	39.9	0.438	0.304^{+}	0.792^{+}
1000d10c	GPGC	11.60	10.1	15.0	2.132	1.704	0.980
	AIC	12.60^{+}	9.7	14.9	2.126	1.784^{+}	0.987^{+}
	RIC	12.55^{+}	9.7	15.0	2.122	1.776^{+}	0.984
	SIC	12.48^{+}	9.6	15.2	2.115	1.754^{+}	0.978
1000d20c	GPGC	9.22	23.1	12.0	1.539	1.325	0.834
	AIC	11.48^{+}	19.6	12.4^{+}	1.575	1.511^{+}	0.810
	RIC	11.37^{+}	19.5	12.2	1.581^{-}	1.533^{+}	0.790
	SIC	10.96^{+}	19.4	12.3	1.589^{-}	1.498^{+}	0.804
1000d40c	GPGC	8.48	47.5	14.0	1.376	1.006	0.797
	AIC	10.14^{+}	42.5	14.2	1.387	1.130^{+}	0.804
	RIC	10.01^{+}	42.6	14.2	1.385	1.126^{+}	0.832
	SIC	9.66^{+}	44.1	14.2	1.382	1.086^{+}	0.828
1000d100c	GPGC	7.91	132.5	15.8	1.172	0.761	0.839
	AIC	9.90^{+}	117.2	16.0	1.189^{-}	0.901^{+}	0.916^{+}
	RIC	9.06^{+}	119.9	16.0^{+}	1.186^{-}	0.839^{+}	0.863
	SIC	8.56	124.7	16.0^{+}	1.172	0.797	0.853

Sparsity is either increased (i.e. made worse) or is similar compared to GPGC when a multi-tree approach is used — this suggests that the single tree approach was overly favouring reducing sparsity at the expense of the overall fitness. Another interesting pattern is that the number of clusters (K) found by the multi-tree approaches was always lower than that found by GPGC; given that GPGC tended to over-estimate K, this can be seen as further evidence that using multiple trees improves clustering performance. Furthermore, a smaller K is likely to directly

Table 4.10: Summary of ARI post-hoc analysis findings. For each dataset, all results with a p-value below 0.05 (5% significance level) are shown. "AIC >GPGC" indicates that AIC had a significantly better ARI than GPGC on the given dataset, with a given p-value.

Dataset	Finding	p-value	Finding	p-value	Finding	p-value	Finding	p-value
10d10cG	AIC >GPGC	0.000	AIC >SIC	0.015	RIC >GPGC	0.002	SIC >GPGC	0.034
10d10c	AIC >GPGC	0.002	SIC >GPGC	0.018	RIC >GPGC	0.028		
10d40c	GPGC >AIC	0.048	GPGC >RIC	0.001	SIC >RIC	0.047		
50d10c	AIC >GPGC	0.000	AIC >RIC	0.040	AIC >SIC	0.003	RIC >GPGC	0.016
50d40c	AIC >GPGC	0.003	AIC >RIC	0.003				
100d40c	AIC >GPGC	0.013	RIC >GPGC	0.006	SIC >GPGC	0.004		
1000d100c	AIC >GPGC	0.000	AIC >RIC	0.037	AIC >SIC	0.004		

improve connectedness as more instances will have neighbours in the same cluster, and separation since having fewer clusters increases the average distance between neighbouring clusters.

In terms of the ARI, the multi-tree approaches were significantly better than GPGC on a number of elliptically-generated datasets, with the RIC, SIC, and AIC methods being significantly better on three, three, and six datasets respectively. Both the AIC and RIC methods have significantly better fitness than GPGC on these datasets, while SIC is not significantly better on 100d10c or 1000d100c.

To better understand which of the three multi-tree methods has the highest performance, we analysed the ARI results further as these give the best overall evaluation of how the multi-tree methods compare to the gold standard. We performed a Kruskal-Wallis rank sum test (at a 5% significance level) followed by post-hoc pair-wise analysis using Dunn's test. The summary of this testing is shown in Table 4.10.

According to the post-hoc analysis, AIC outperformed GPGC six times, whereas RIC and SIC outperformed GPGC four and three times respectively. AIC outperformed RIC and SIC in three cases each as well. In one case, SIC outperformed SIC. Furthermore, AIC generally had a smaller p-value when it outperformed GPGC compared to SIC and RIC. Based on these findings, we conclude that AIC is the most effective of the three proposed multi-tree approaches. This may be because AIC is a

more "aggressive" form of crossover, which enables more knowledge to be transferred between different GP individuals during the same number of generations compared to RIC or SIC. Based on this, we use GPGC-AIC in the next section to compare to other clustering methods.

4.9.2 GPGC-AIC compared to the Benchmarks

Tables 4.11 and 4.12 show how the proposed GPGC-AIC method compares to the six benchmarks across the datasets tested. For each of the seven methods, we provide the (mean) number of clusters (K), as well as the same four metrics of cluster quality as before. Note that k-means++ requires K to be pre-defined, and so always obtains the correct K value. We use the two-tailed Mann Whitney U-test as in Section 4.9.1: a "$+$" indicates that a baseline method is significantly better than the GPGC-AIC method, a "$-$" indicates it is significantly worse, and no symbol indicates no significant difference is found.

Table 4.11 shows the results on the datasets that were generated using a Gaussian distribution. In terms of the ARI, the AIC method is significantly worse than either the MCL or MOCK method across the three datasets, but generally outperforms all the other baselines. As these datasets were generated with a Gaussian distribution, they tend to contain very well-formed hyper-spherical clusters, and so methods such as MCL are very effective at clustering these correctly.

On the datasets generated using an elliptical distribution, shown in Table 4.12, GPGC is significantly better than all baselines excluding MOCK across all datasets containing 10 features (10d*c). While the MOCK method is competitive (or better) on the datasets with 10 and 20 clusters, it achieves a very poor ARI on the more difficult datasets with 40 and 100 clusters, where the AIC method is clearly superior. The remaining baseline methods are nearly always significantly worse than AIC on these datasets. The NG baselines are particularly inconsistent, with the number of clusters and ARI values varying by up to three times depending on the number of nearest neighbours chosen. AIC can also

Table 4.11: Baselines: Datasets using a Gaussian distribution.

Dataset	Method	K	Conn.	Spar.*	Sep.	ARI
10d10cGaussian	AIC	8.8	51.4	0.324	0.154	0.880
	k-means++	10.0	50.4^{-}	0.341^{-}	0.133^{-}	0.848^{-}
	MCL	8.0	52.8^{+}	0.323	0.137^{-}	0.910
	MOCK	13.6	41.4^{-}	0.291^{-}	0.167^{+}	0.963^{+}
	NG-2NN	4.0	45.3^{-}	0.317^{-}	0.193^{+}	0.368^{-}
	NG-3NN	1.0	57.6^{+}	0.428^{-}	0.000^{-}	0.248^{-}
	OPT-0.005	39.0	27.9^{-}	0.227^{-}	0.103^{-}	0.572^{-}
10d20cGaussian	AIC	19.5	47.3	0.268	0.377	0.980
	k-means++	20.0	43.8^{-}	0.273	0.293^{-}	0.872^{-}
	MCL	20.0	47.2^{-}	0.269^{-}	0.373^{-}	0.998^{+}
	MOCK	20.7	45.7^{-}	0.265^{-}	0.359^{-}	0.990
	NG-2NN	19.0	47.3^{+}	0.268	0.381^{+}	0.965^{-}
	NG-3NN	19.0	47.3^{+}	0.268	0.381^{+}	0.965^{-}
	OPT-0.001	26.0	38.2^{-}	0.287^{-}	0.218^{-}	0.895^{-}
10d40cGaussian	AIC	33.7	48.9	0.265	0.337	0.943
	k-means++	40.0	44.5^{-}	0.270^{-}	0.260^{-}	0.895^{-}
	MCL	40.0	47.2^{-}	0.264^{-}	0.335^{-}	0.999^{+}
	MOCK	38.0	46.4^{-}	0.261^{-}	0.321^{-}	0.960^{+}
	NG-2NN	40.0	46.3^{-}	0.261^{-}	0.332^{-}	0.984^{+}
	NG-3NN	37.0	47.6^{-}	0.263^{-}	0.342^{+}	0.951
	OPT-0.001	55.0	36.3^{-}	0.298^{-}	0.175^{-}	0.850^{-}

automatically find the number of clusters much more accurately than OPTICS, and produces less sparse and more separated clusters than k-means++ across these datasets. In contrast to the previous datasets, the MCL method struggles significantly with these non-hyper-spherical datasets — a pattern that is also true for the remaining datasets and that highlights a key weakness with the MCL method. Similar patterns are seen across the 50d*c datasets, with the exception being on 50d10c where NG-3NN achieves a near perfect result. On the high-dimensional datasets (100d*c, 1000d*c), the MOCK method has a high variance in accuracy, with ARI values ranging from 0.434 to 0.897. In contrast, the AIC method achieves consistently good performance, with the lowest ARI achieved being 0.771 and the highest being 0.998. While the MOCK method is superior on 1000d40c, its inconsistency on the other datasets makes it harder to use confidently in practice. On this set of datasets, the NG baselines achieve better results than previously in terms of the ARI,

Table 4.12: Baselines: Datasets using an Elliptical Distribution (Part 1).

Dataset	Method	K	Conn.	Spar.*	Sep.	ARI
10d10c	AIC	24.0	71.7	0.166	0.059	0.814
	k-means++	10.0	85.5^{+}	0.212^{-}	0.038^{-}	0.552^{-}
	MCL	15.0	83.3^{+}	0.205^{-}	0.050^{-}	0.703^{-}
	MOCK	17.8	86.1^{+}	0.180^{-}	0.065	0.793
	NG-2NN	9.0	75.0	0.176^{-}	0.071^{+}	0.510^{-}
	NG-3NN	5.0	82.3^{+}	0.187^{-}	0.085^{+}	0.323^{-}
	OPT-0.05	32.0	84.4^{+}	0.063^{-}	0.025^{-}	0.239^{-}
10d20c	‘ AIC	22.5	77.4	0.154	0.106	0.666
	k-means++	20.0	73.3^{-}	0.190^{-}	0.087^{-}	0.459^{-}
	MCL	28.0	69.4^{-}	0.173^{-}	0.095^{-}	0.451^{-}
	MOCK	27.5	79.3^{+}	0.152	0.113^{+}	0.752^{+}
	NG-2NN	36.0	69.5^{-}	0.131^{-}	0.094^{-}	0.584^{-}
	NG-3NN	16.0	83.1^{+}	0.145^{-}	0.110^{+}	0.355^{-}
	OPT-0.001	69.0	45.9^{-}	0.210^{-}	0.053^{-}	0.344^{-}
10d40c	AIC	49.2	76.6	0.139	0.072	0.522
	k-means++	40.0	77.2	0.169^{-}	0.070	0.417^{-}
	MCL	54.0	69.2^{-}	0.149^{-}	0.094^{+}	0.288^{-}
	MOCK	28.8	84.2^{+}	0.146^{-}	0.087^{+}	0.232^{-}
	NG-2NN	43.0	68.4^{-}	0.110^{-}	0.085^{+}	0.256^{-}
	NG-3NN	13.0	73.6^{-}	0.131^{-}	0.107^{+}	0.082^{-}
	OPT-0.005	93.0	61.4^{-}	0.153^{-}	0.048^{-}	0.299^{-}
10d100c	AIC	106.4	75.6	0.131	0.066	0.421
	k-means++	100.0	80.4^{+}	0.153^{-}	0.056^{-}	0.398
	MCL	98.0	77.3^{+}	0.136^{-}	0.070^{+}	0.125^{-}
	MOCK	62.0	86.2^{+}	0.137^{-}	0.074^{+}	0.087^{-}
	NG-2NN	91.0	70.6^{-}	0.108^{-}	0.080^{+}	0.049^{-}
	NG-3NN	26.0	73.4	0.096^{-}	0.084^{+}	0.030^{-}
	OPT-0.01	197.0	69.5^{-}	0.091^{-}	0.042^{-}	0.054^{-}
50d10c	AIC	10.0	59.5	0.472	0.341	0.987
	k-means++	10.0	52.2^{-}	0.555^{-}	0.102^{-}	0.485^{-}
	MCL	12.0	54.1^{-}	0.519^{-}	0.102^{-}	0.604^{-}
	MOCK	14.4	56.7^{-}	0.494^{-}	0.217^{-}	0.811^{-}
	NG-2NN	18.0	53.6^{-}	0.372^{-}	0.168^{-}	0.967^{-}
	NG-3NN	11.0	58.8^{-}	0.456^{-}	0.302^{-}	0.999^{+}
	OPT-0.05	28.0	67.7^{+}	0.196^{-}	0.081^{-}	0.369^{-}
50d20c	AIC	21.1	51.8	0.359	0.273	0.837
	k-means++	20.0	44.3^{-}	0.429^{-}	0.168^{-}	0.353^{-}
	MCL	26.0	42.0^{-}	0.415^{-}	0.175^{-}	0.482^{-}
	MOCK	24.3	50.3^{-}	0.375^{-}	0.273	0.884
	NG-2NN	44.0	41.0^{-}	0.304^{-}	0.213^{-}	0.831
	NG-3NN	23.0	49.0^{-}	0.369^{-}	0.286	0.808^{-}
	OPT-0.005	73.0	32.2^{-}	0.455^{-}	0.129^{-}	0.386^{-}
50d40c	AIC	45.2	56.1	0.308	0.220	0.810
	k-means++	40.0	50.7^{-}	0.352^{-}	0.140^{-}	0.254^{-}
	MCL	48.0	48.6^{-}	0.329^{-}	0.161^{-}	0.351^{-}
	MOCK	42.7	56.6^{+}	0.315^{-}	0.253^{+}	0.867
	NG-2NN	86.0	50.1^{-}	0.249^{-}	0.170^{-}	0.762^{-}
	NG-3NN	46.0	56.3	0.279^{-}	0.217	0.738^{-}
	OPT-0.05	73.0	53.5^{-}	0.239^{-}	0.098^{-}	0.163^{-}

Table 4.12: Baselines: Datasets using an Elliptical Distribution (Part 2).

Dataset	Method	K	Conn.	Spar.*	Sep.	ARI
100d10c	AIC	9.8	48.1	0.621	0.580	0.998
	k-means++	10.0	45.5^{-}	0.695^{-}	0.131^{-}	0.562^{-}
	MCL	16.0	40.5^{-}	0.677^{-}	0.207^{-}	0.877^{-}
	MOCK	28.8	45.7^{-}	0.590^{-}	0.170^{-}	0.548^{-}
	NG-2NN	16.0	44.6^{-}	0.552^{-}	0.287^{-}	0.934^{-}
	NG-3NN	11.0	45.8^{-}	0.647^{-}	0.513^{-}	0.989^{-}
	OPT-0.001	92.0	37.6^{-}	0.372^{-}	0.113^{-}	0.455^{-}
100d20c	AIC	20.6	38.2	0.535	0.494	0.917
	k-means++	20.0	34.1^{-}	0.595^{-}	0.232^{-}	0.374^{-}
	MCL	27.0	29.5^{-}	0.594^{-}	0.283^{-}	0.587^{-}
	MOCK	24.6	35.7^{-}	0.573^{-}	0.487	0.897
	NG-2NN	41.0	32.1^{-}	0.450^{-}	0.291^{-}	0.819^{-}
	NG-3NN	25.0	34.7^{-}	0.529	0.520^{+}	0.965^{+}
	OPT-0.01	76.0	28.1^{-}	0.505^{-}	0.168^{-}	0.368^{-}
100d40c	AIC	45.9	40.6	0.440	0.316	0.771
	k-means++	40.0	35.3^{-}	0.492^{-}	0.226^{-}	0.268^{-}
	MCL	57.0	32.9^{-}	0.473^{-}	0.265^{-}	0.467^{-}
	MOCK	41.8	39.4^{-}	0.476^{-}	0.371^{+}	0.784
	NG-2NN	91.0	34.6^{-}	0.375^{-}	0.299^{-}	0.791
	NG-3NN	49.0	36.1^{-}	0.431	0.393^{+}	0.711^{-}
	OPT-0.001	140.0	25.6^{-}	0.593^{-}	0.150^{-}	0.430^{-}
1000d10c	AIC	9.7	14.9	2.126	1.784	0.987
	k-means++	10.0	13.8^{-}	2.347^{-}	0.385^{-}	0.488^{-}
	MCL	10.0	12.2^{-}	2.407^{-}	0.445^{-}	0.474^{-}
	MOCK	16.0	14.5^{-}	2.079^{-}	0.995^{-}	0.800^{-}
	NG-2NN	21.0	14.9^{+}	1.681^{-}	0.610^{-}	0.932^{-}
	NG-3NN	10.0	15.5^{+}	2.078^{-}	1.541^{-}	0.947^{-}
	OPT-0.005	86.0	13.1^{-}	1.138^{-}	0.455^{-}	0.343^{-}
1000d20c	AIC	19.6	12.4	1.575	1.511	0.810
	k-means++	20.0	9.7^{-}	1.885^{-}	0.832^{-}	0.376^{-}
	MCL	24.0	9.3^{-}	1.828^{-}	0.748^{-}	0.339^{-}
	MOCK	25.5	10.8^{-}	1.706^{-}	1.423^{-}	0.896
	NG-2NN	47.0	9.8^{-}	1.409^{-}	1.014^{-}	0.736^{-}
	NG-3NN	26.0	10.5^{-}	1.531^{-}	1.379^{-}	0.945^{+}
	OPT-0.001	67.0	7.0^{-}	2.142^{-}	0.589^{-}	0.453^{-}
1000d40c	AIC	42.5	14.2	1.387	1.130	0.804
	k-means++	40.0	11.7^{-}	1.556^{-}	0.632^{-}	0.219^{-}
	MCL	47.0	10.8^{-}	1.500^{-}	0.676^{-}	0.157^{-}
	MOCK	41.7	13.6^{-}	1.488^{-}	1.231^{+}	0.887^{+}
	NG-2NN	94.0	12.1^{-}	1.209^{-}	0.846^{-}	0.740^{-}
	NG-3NN	52.0	13.7^{-}	1.346^{-}	1.130	0.898^{+}
	OPT-0.001	132.0	9.3^{-}	1.841^{-}	0.482^{-}	0.422^{-}
1000d100c	AIC	117.2	16.0	1.189	0.901	0.916
	k-means++	100.0	14.5^{-}	1.270^{-}	0.523^{-}	0.103^{-}
	MCL	204.0	11.5^{-}	1.242^{-}	0.482^{-}	0.189^{-}
	MOCK	64.3	16.7^{+}	1.265^{-}	1.146^{+}	0.434^{-}
	NG-2NN	229.0	14.7^{-}	1.014^{-}	0.693^{-}	0.761^{-}
	NG-3NN	132.0	16.0^{-}	1.107^{-}	0.933	0.863^{-}
	OPT-0.05	182.0	16.0	0.917^{-}	0.414^{-}	0.106^{-}

Table 4.13: Number of wins of each algorithm across the 17 datasets.

AIC	k-means++	MCL	MOCK	NG-2NN	NG-3NN	OPTICS
6	0	2	4	1	4	0

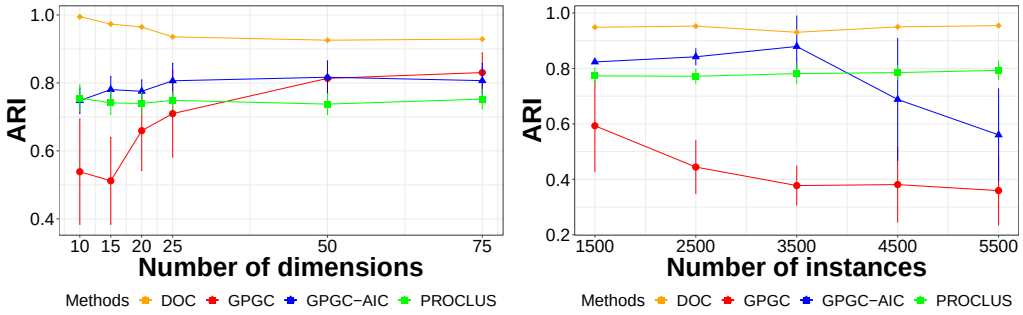
but GPGC is only ever significantly worse than one of NG-2NN and NG-3NN at most. GPGC also often has significantly better connectedness and separation than one or both of the NG baselines, suggesting it is a more consistent choice given that it is difficult to determine the number of nearest neighbours in advance (as the NG methods assume). All of the graph-based approaches are superior to k-means++ (due to the non-hyper-spherical cluster shape), and OPTICS across these datasets. The AIC method also appears to be the method that predicts K most accurately overall across these datasets, especially where $K = 100$.

Summary

Table 4.13 shows the number of datasets for which each clustering method was the winner (i.e. highest mean ARI). The AIC method was the most successful, with six wins compared to four for the closest methods (NG-3NN and MOCK). This is consistent with our previous analysis, which showed AIC was the most consistent and best-performing method across datasets with high dimensionality, and was competitive with MOCK on the remaining datasets. The remaining baselines, with the exception of MCL, were almost always outperformed by at least one of AIC and MOCK. Given that MOCK is a multi-objective approach, we are hopeful that a future multi-objective variation of AIC would be able to improve the GPGC method even further and allow it to achieve a higher number of wins.

4.9.3 Subspace Clustering

The fitness function proposed in this section is designed to use all features in the feature space when calculating distances, as our primary



(a) Varying the number of dimensions. (b) Varying the number of instances.

Figure 4.8: ARI achieved by each clustering method on the OpenSubspace clustering datasets, where either the number of dimensions, or number of instances are varied.

goal is to reduce dimensionality and produce interpretable similarity functions, while maintaining good cluster quality. However, a natural extension of our proposed approach is to apply it to subspace clustering problems, as our GP representation has the potential to use different feature subsets in different clusters. To investigate the plausibility of this extension, we applied GPGC and GPGC-AIC to datasets from OpenSubspace [116], a collection of popular subspace benchmarking datasets. We chose to use the PROjected CLUStering algorithm (PROCLUS) [2] and Density-based Optimal projective Clustering (DOC) algorithm [137] for comparison, as examples of commonly used cell-based and clustering-oriented subspace clustering algorithms respectively. We chose PROCLUS and DOC as they have been shown to have superior (or similar) performance to other subspace algorithms in their paradigm [116]. We do not compare to a clustering algorithm from the third paradigm of density-based subspace clustering, as these methods all produced overlapping (i.e. non-crisp) clusters, which is not the focus of this thesis.

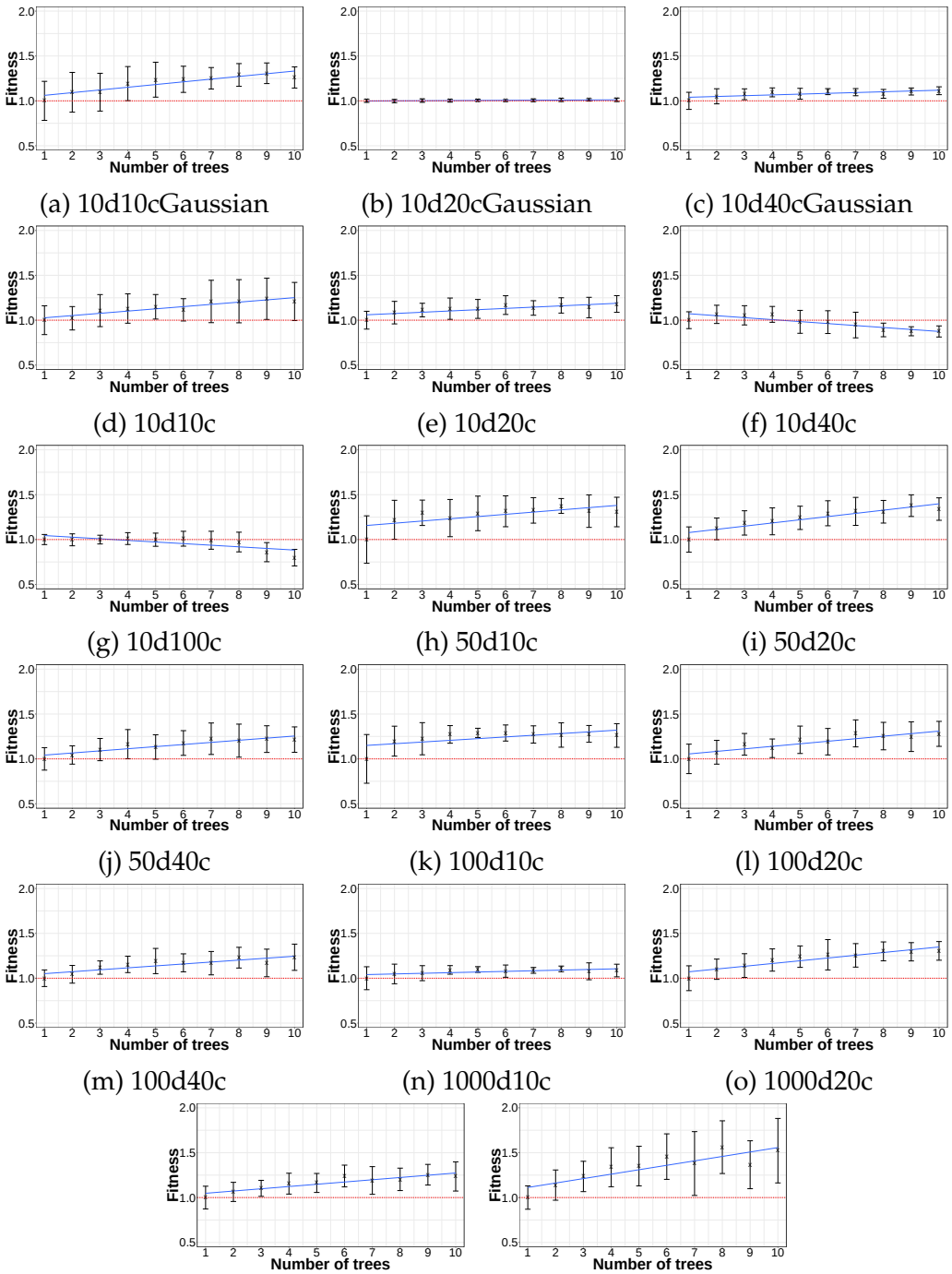
Figure 4.8 shows the performance of each of the four methods (GPGC, GPGC-AIC, PROCLUS, and DOC) as the number of dimensions is varied (Figure 4.8a) and the number of instances (Figure 4.8b) is varied respectively. For each dataset, we plot the ARI achieved by each method,

as well as the standard deviation across 30 runs (the vertical bars). The two proposed methods are competitive with PROCLUS as the dimensionality is increased, but PROCLUS clearly outperforms GPGC and is slightly better than GPGC-AIC as the number of instances is increased. The DOC method is clearly the best of all the methods on both categories of datasets. However, GPGC-AIC in particular shows some promise given it has not been optimised for this task, but can still achieve competitive results often with one common subspace clustering method. Furthermore, GPGC-AIC is clearly superior to vanilla GPGC, which further reinforces our previous findings. We hope to further improve GPGC-AIC and make it a competitive subspace clustering algorithm in the future by developing a new fitness function and designing new genetic operators.

4.9.4 Number of Trees: Effect on Fitness

The number of trees to use in a multi-tree approach can be considered to be a form of parameter tuning. To investigate the effect of different numbers of trees on the training performance of the proposed multi-tree approach, we tested a range of trees for $t \in [1, 10]$, using the AIC crossover method. We limit t to a maximum of 10, as we found that multi-tree GP did not have improved performance, and trained more slowly at higher t values. Note that the $t = 1$ case is not equivalent to the single-tree GPGC method, due to the smaller maximum program depth of 5. For each dataset, we calculated the mean fitness of the 30 runs for each value of t. The results are plotted in Figure 4.9. Each plot corresponds to a single dataset, with a red dotted line indicating the baseline performance where $t = 1$, and each point corresponds to the *relative* fitness for a given value of t. The error bars show the standard deviation of each point, for the 30 runs performed for that t value. The blue solid line is a trend-line fitted to the 10 points.

On the majority of the plots (14 out of 17), increasing the number of trees causes an increase in the fitness obtained; the 10d20cGaussian plot has no noticeable improvement as t is increased, while on the 10d40c and



(a) 10d10cGaussian (b) 10d20cGaussian (c) 10d40cGaussian

(d) 10d10c (e) 10d20c (f) 10d40c

(g) 10d100c (h) 50d10c (i) 50d20c

(j) 50d40c (k) 100d10c (l) 100d20c

(m) 100d40c (n) 1000d10c (o) 1000d20c

(p) 1000d40c (q) 1000d100c

Figure 4.9: Effect on training performance as the number of trees is increased.

10d100c plots, the fitness is actually reduced by using more trees. This is consistent with the results presented in Table 4.9, where the AIC method had significantly worse fitness on the 10d40c dataset, and was not significantly different on the 10d100c dataset — on the 10d100c plot, the fitness value for $t = 7$ is very close to the baseline (i.e. a relative fitness of 1). For the 10d20cGaussian plot, we hypothesise that there is little room for fitness improvement as t is increased, as shown by the small changes in fitness and ARI performance compared to GPGC in Table 4.8. On the 14 plots where there was a positive association, the fitness improvement is between 10% (on 1000d10c), and slightly over 50% (on 1000d100c), with improvements of around 25% on the majority of the datasets.

The optimal value of t varies depending on the dataset that is used — however, fitness tends to peak at a certain value of t for each dataset, before remaining relatively constant or dipping slightly (except for those where t does not improve performance). The best value of t for each dataset is hence the lowest value of t for which performance is significantly better than all lower values of t, as this provides the best balance of maximising fitness while maintaining the interpretability and computational benefits of a lower t value. For most datasets, this value is between $t = 5$ and $t = 8$, except for 50d20c and 1000d10c, where a t value of 9 and 4 seem to be the best, respectively. Hence, we suggest a value of $t = 7$ or $t = 8$ may be used as a good starting point for achieving good fitness on other datasets.

4.10 Evolving Similarity Functions: Further Analysis

4.10.1 Evolved GP Trees

In addition to achieving good clustering performance, our proposed methods are also expected to automatically select a subset of features and construct new, more powerful high-level features due to the tree-based

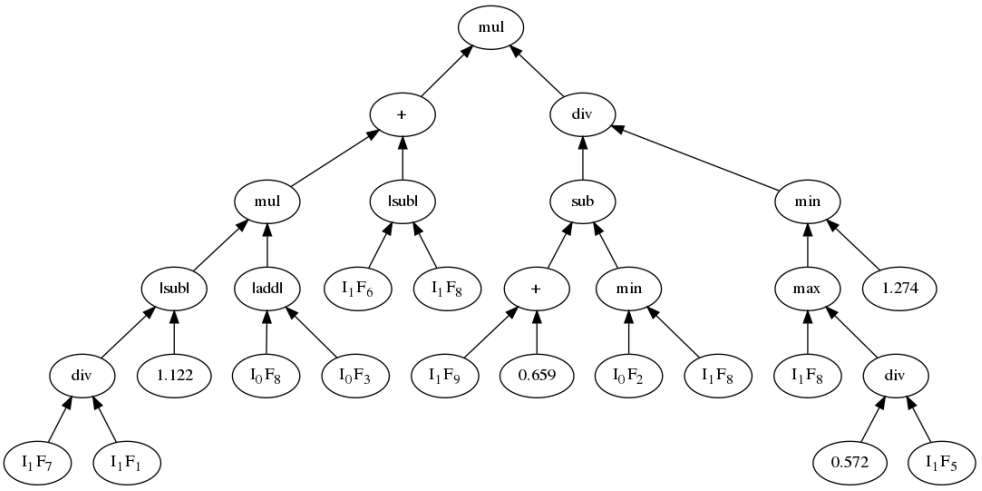


Figure 4.10: An evolved individual on the 10d10c dataset using the single-tree approach. Individual has a fitness of 21.37 and ARI of 0.9144 and produces $K = 22$ clusters.

GP structure used. To evaluate the feature manipulation performance of our proposed methods, we analyse an example evolved individual for both the single- and multi-tree approaches in this subsection.

Figure 4.10 shows a single-tree individual evolved by GPGC on the 10d10c dataset, which has a very good ARI result of 0.9144. We can see that the tree produced is able to combine a number of different sub-trees to effectively construct a custom similarity function that can vary its behaviour across the dataset through the use of conditional *max* and *min* operators. A range of *building blocks* are used to find the similarity of two instances, from simple feature weighting operations (e.g. $0.572 \div I_1F_5$, $I_1F_9 + 0.659$) to more advanced feature comparisons (e.g. $min(I_0F_2, I_1F_8)$), with high-level features formed by combining these building blocks in a variety of ways. By evolving a similarity function tailored to this dataset, GPGC is able to outperform the benchmark methods, which use the inflexible Euclidean distance function. This evolved function gives a different nearest neighbour to that of Euclidean distance for 97.04% of the instances, and on average chooses the 5.4th nearest neighbour according to Euclidean distance ordering. Clearly,

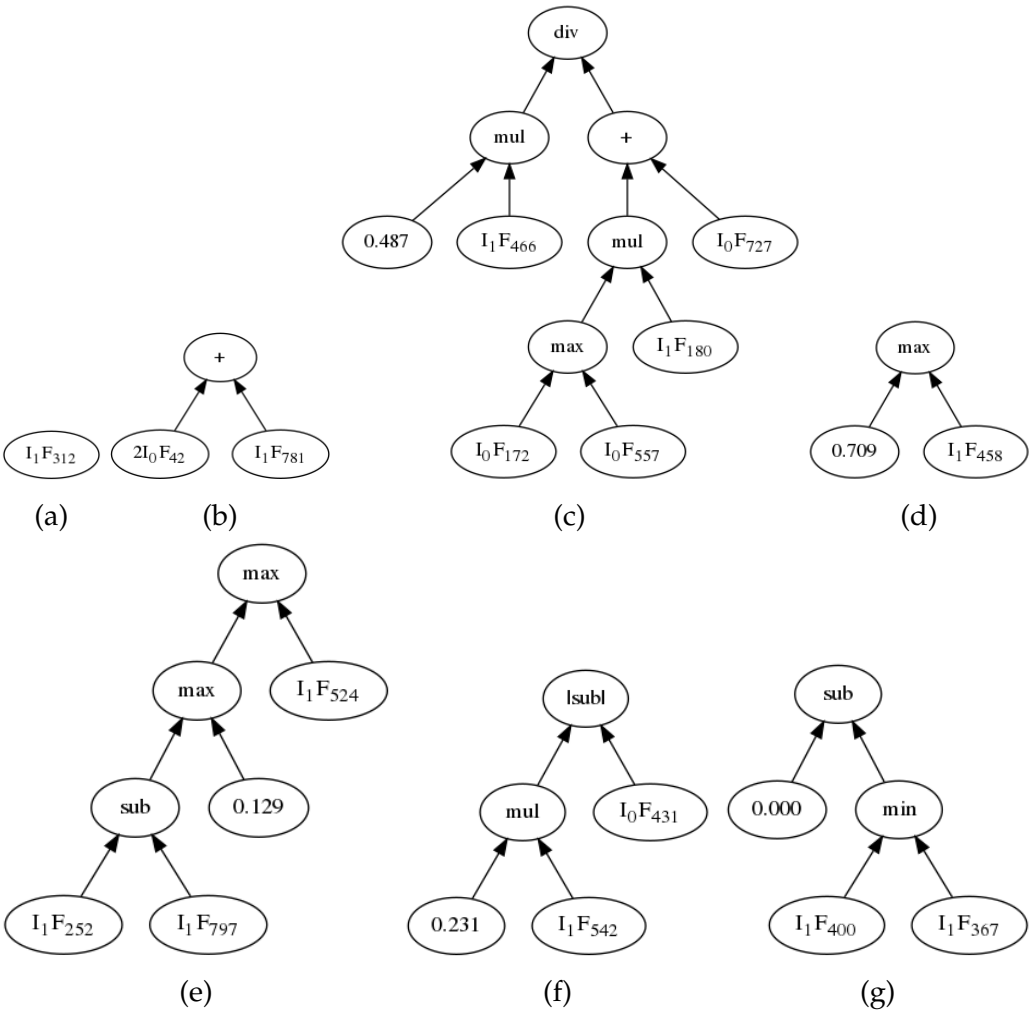


Figure 4.11: An evolved individual on the 1000d20c dataset using AIC crossover and $t = 7$. Individual has a fitness of 10.43 and ARI of 0.9616 and produces $K = 22$ clusters.

GPGC has produced a significantly different ordering that is more appropriate for this dataset than normal Euclidean distance ordering.

Figure 4.11 shows an example of a multi-tree individual evolved on the 1000d20c dataset, which has a very good ARI result of 0.9616. We have simplified the trees where appropriate to aid interpretability by computing constants and removing dead branches. The seven simplified evolved trees are generally quite simple, with only one tree (c) being the maximum depth of five; and one, two, two, and one trees having depths

of four (e), three (f,g), two (b,d), and one (a), respectively. While it is difficult to fully understand why these trees perform well across each instance in the dataset, it is possible to gain insight by examining the general behaviour of each tree ((a)–(g)), as shown below:

(a) simple feature selection of I_1F_{312}.

(b) computing a weighted sum of two selected features.

(c) constructing a more powerful high-level feature by weighting and constructing non-linear combinations of five original features.

(d) thresholding I_1F_{458} so that it has a minor impact on the total similarity.

(e) finding the maximum of: a feature, a constant value, and the difference between two features. This gives varied behaviour based on the instances being considered.

(f) finding the absolute difference between two features, with one feature scaled.

(g) finds the maximum of two features, and then takes the result as a negative. In this way, the bigger the result, the less similar the two instances are said to be.

Each of the seven trees evaluated above has distinctive and interesting behaviour, which gives insight into which features are useful in the dataset, and into what relationships between features can be used to gauge instances' similarities accurately. In contrast, a standard distance function cannot provide such insight, as it uses the *full* feature set and performs only linear comparisons between instances' features. Of the 1000 features in the 1000d20c dataset, the example individual in Figure 4.11 uses only 15 features to build its seven similarity functions. This means the clustering partition produced is much more interpretable than one produced by a standard nearest-neighbour graph-based clustering algorithm. This evolved meta-similarity function chooses the

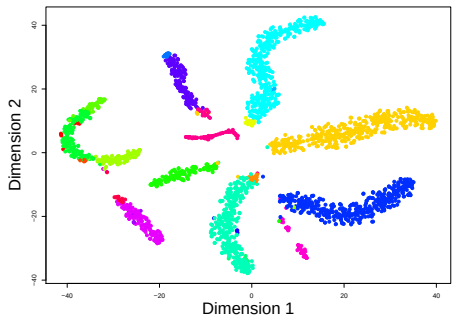
same nearest neighbour as that of Euclidean distance on only 2.67% of the instances in the dataset. On average, the 6.2th nearest neighbour is chosen as the first nearest neighbour: this is a similar trend to that of the previous example in that the neighbours have been significantly re-ordered to be tailored for this dataset.

4.10.2 Visualising the Clusters Found

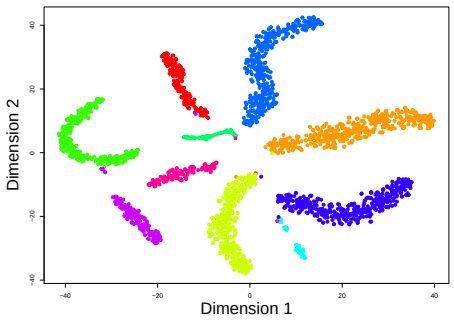
To further analyse the examples discussed in Section 4.10.1, we visualise the clusters produced compared to the ground truth clustering in this subsection using the commonly used t-SNE visualisation method [171], which minimises the probability distribution divergence between the two-dimensional visualisation and the original feature space.

Figure 4.12 shows the clusters produced by GPGC, k-means++ and the NG-2NN methods on the 10d10c dataset. For GPGC, we use the same evolved individual as in Section 4.10.1. For k-means++, we chose the result with the highest ARI of the 30 runs. NG-2NN is deterministic, and so the single result is shown. In addition, the ground truth is shown in Figure 4.12 for reference. It is clear that GPGC is able to most accurately reproduce the ground truth, with the majority of the clusters mapping to the ground truth well, asides from a few instances in each cluster. The exception is on the horseshoe-shaped cluster on the left of the visualisation, where GPGC has over-clustered the data by splitting this cluster into two. The k-means++ method clearly performs very poorly, with only the horseshoe-shaped cluster being clustered nearly correctly; all other clusters have significant overlap. The NG-2NN method also produces clearly incorrect clusters, with many clusters being combined, including four distinct clusters combined into one single blue cluster. GPGC is clearly able to better find the natural clusters compared to these baseline methods.

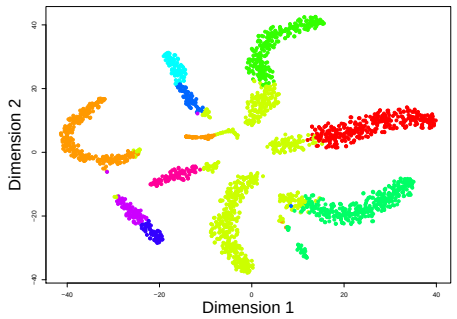
Figure 4.13 (a) and (b) show the clusters produced by the GPGC-MT method (using the evolved tree discussed in Section 4.10.1) and the ground truth respectively on the 1000d20c dataset. GPGC-MT



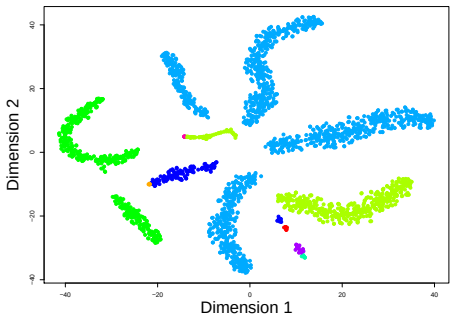
(a) GPGC (F-M: 0.9144).



(b) Ground Truth.



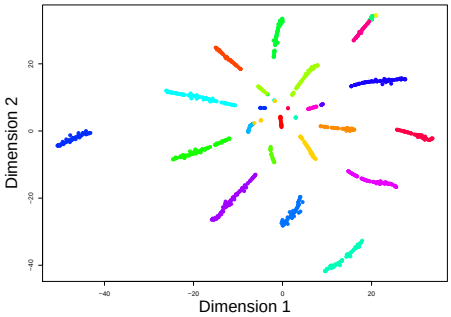
(c) k-means++ (F-M: 0.6134.)



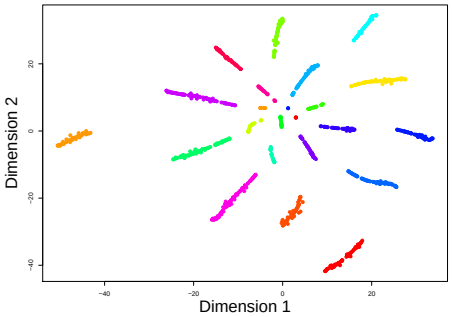
(d) NG-2NN.

Figure 4.12: Visualising the partitions chosen by a GP individual compared to a sample of the baseline methods on the 10d10c dataset. t-SNE [171] is used to reduce dimensionality to two dimensions. Each colour corresponds to a single cluster.

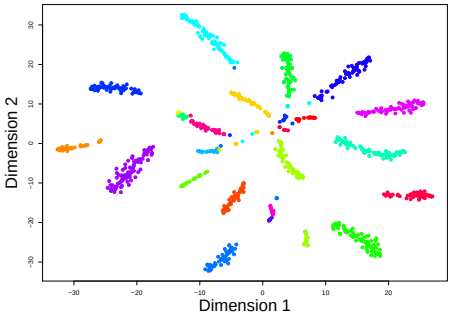
reproduces the ground truth accurately, with only a small amount of over-clustering. The GPGC-MT method uses only a subset of features in the evolved trees; in this case, only 15 of the 1000 features are used. To analyse whether using so few features would reduce the interpretability of the clusters produced, we performed another set of visualisations that used only the 15 selected features as input, as shown in Figure 4.13 (c) and (d). The clusters shown in these visualisations are still very distinct and well-separated, which suggests that the GPGC-MT method was able to successfully perform feature selection implicitly in the evolved similarity functions. While t-SNE is able to reliably project the feature space into two dimensions, it does so at the cost of interpretability – the two dimensions produced cannot be easily mapped back to the original



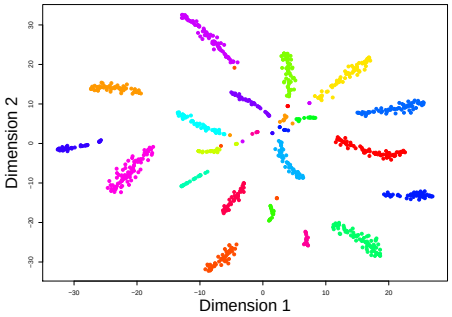
(a) All features: GPGC-MT.



(b) All features: Ground Truth.



(c) Selected features: GPGC-MT.



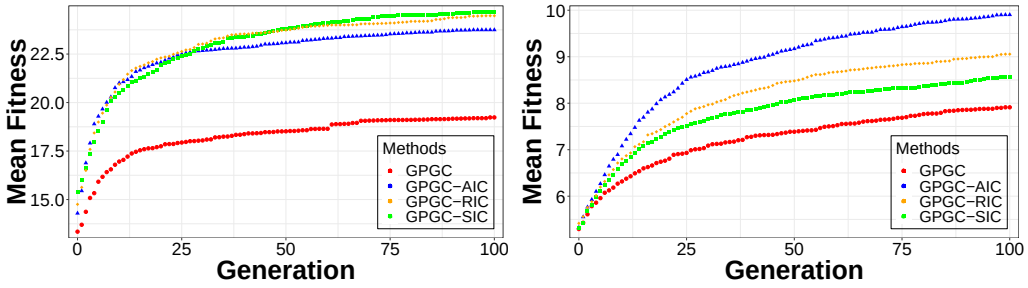
(d) Selected features: Ground Truth.

Figure 4.13: Visualising the partitions chosen by a GP individual on the 1000d20c dataset. t-SNE [171] is used to reduce dimensionality to two dimensions. Figures (a) and (b) show the visualisations formed when all features are used by t-SNE, whereas (c) and (d) show the visualisations using only the features used in the GP tree. Each colour corresponds to a single cluster.

feature set, and so it is very difficult to analyse why a cluster contains certain instances. In contrast, GPGC-MT uses only a small subset of the feature set, and explicitly combines features in an interpretable manner in the evolved trees.

4.10.3 Evolutionary Process

To further analyse the learning effectiveness of the proposed methods (GPGC and the three multi-tree crossover approaches), we plot the fitness over the evolutionary process for the 10d10cGaussian and 1000d100c datasets, as shown in Figure 4.14 (a) and (b) respectively. For



(a) 10d10cGaussian. (b) 1000d100c.

Figure 4.14: Fitness of the four proposed methods over the evolutionary process.

each dataset, we plot the mean fitness of the best individual at each generation, taken across the 30 independent runs. These datasets were selected as they represent the datasets with the lowest and highest m and K values, and are from each of the two different generators used. Both datasets show the same pattern for the single-tree compared with the multi-tree approaches: while all methods begin at similar fitnesses, the multi-tree methods increase in mean fitness at a significantly faster rate, and reaches a much higher mean fitness overall. Indeed, the final fitness of GPGC at the 100th generation is achieved by each of the multi-tree approaches by generation 25 in both datasets. It is clear that the multi-tree approaches can train more efficiently (i.e. with a steeper initial slope), and effectively, by reaching a higher final fitness over the same number of generations. While the GPGC-AIC method is slightly outperformed by the other two crossover approaches on the 10d10cGaussian dataset, it is clearly the best method on the more difficult 1000d100c dataset, which reinforces our view that this is the best of the proposed approaches. While fitness appears to have levelled off by generation 100 on the 10d10cGaussian dataset, it appears that additional generations could improve the performance on the 1000d100c dataset even further.

4.11 Chapter Conclusions

This chapter introduced two new approaches to improving the performance of clustering algorithms by using GP for FC.

The first method used a wrapper-based approach, which has proven success in the supervised domain, to improve the clustering accuracy of the archetypal k-means clustering algorithm. Two representations were proposed, one using a multi-tree approach and the other using a vector approach. In addition, two potential fitness functions that could be used for training high performing GP trees were explored. Both representations and fitness functions were shown to significantly improve performance compared to the base k-means algorithm across a range of real-world and difficult synthetic datasets. A number of evolved GP trees were analysed and shown to perform effective and efficient FC even in a very small tree.

The second method (GPGC) used a more creative approach to employing FC in clustering, by automatically evolving similarity functions tailored to the clustering algorithm used and dataset being analysed. This is a particularly novel application of GP for FC as it represents the first time that similarity functions have been automatically evolved to be tailored to a specific problem. There is no similar approach in supervised learning, as similarity functions are not heavily used in supervised domains. The results of our experiments showed that the automatically generated similarity functions could improve the performance and consistency of clustering algorithms using a graph representation, while producing more interpretable similarity metrics, which have the potential to be understood by a domain expert as they select only the most important features in a dataset. We also showed that a multi-tree GP approach could be utilised to further improve the performance by automatically evolving several highly-specific similarity functions, which are able to specialise on different components of the overall clustering problem.

While these two methods represent two distinct approaches to GP for FC in clustering, they both provide a number of common conclusions. The use of GP for FC in clustering clearly significantly reduced the complexity of the clustering models formed: the first method explicitly creates a small set of sophisticated features, whereas GPGC creates more concise and specific similarity functions that distinguish clusters more accurately while containing fewer terms than traditional distance metrics. This transforms the search space of the clustering algorithm being used into a more appropriate one, which improves the performance of the algorithm and prevents it from becoming stuck in local optima. Perhaps more uniquely though, this decrease in model complexity has tangible benefits for the interpretability of the clustering solutions. As clustering is often performed as a data exploration task, it is important not only that good clusters are found, but also that the data scientist can understand *how* these clusters are formed from the original features. Using fewer features combined in a sophisticated manner allows for better understanding in this regard.

This chapter showcased how GP-based FC could be used to improve the clustering performance and interpretability of models in clustering problems. While clustering is the most well-known unsupervised learning task, there are many other tasks that could benefit from the use of GP-based FC. The next chapter applies GP-based FC to an important research area that is very underdeveloped: the automatic generation of difficult datasets for benchmarking feature selection algorithms.

Chapter 5

Generating Benchmark Feature Selection Datasets with Genetic Programming

5.1 Introduction

Recently, FS has become an increasingly important area of research due to the surge in high-dimensional datasets in almost all areas of modern life. A plethora of feature selection algorithms have been proposed, but it is difficult to fully and truly analyse the quality of a given algorithm. Ideally, an algorithm would be evaluated by measuring how well it removes known bad features. Acquiring datasets with such features is inherently difficult, and so a common technique is to add synthetic bad features to an existing dataset. While adding noisy features is an easy task, it is very difficult to automatically add complex, redundant features. Very few methods exist for generating such features for use in benchmarking FS algorithms. GP has two key characteristics for this task: it can evolve functional mappings from some inputs (source features) to outputs (redundant features) and it can use a broad range of optimisation criteria, such as those that measure the redundancy between features. Mutual information (MI) is widely used to measure the redundancy

between features, by measuring how much information about instances of the dataset that two features share. There is no known work using GP to automatically evolve redundant features for benchmarking FS algorithms.

5.1.1 Chapter Goals

This chapter aims to propose the first approach to automatically generating redundant features (r.fs) for creating benchmark FS datasets from existing datasets, using GP. This is expected to allow FS algorithms to be compared much more rigorously than currently, by allowing existing datasets to be augmented with known redundant features that are redundant with existing features in the dataset in a variety of non-trivial (non-linear) ways. This chapter will investigate:

- whether a multi-tree GP representation can be used to automatically evolving multiple redundant features from a source feature;
- if a MI-based fitness function can be used to optimise the redundancy between the source and created features;
- whether these created features exhibit a variety of non-trivial redundancies, and if they impact the performance of FS algorithms;
- if a multivariate approach where a set of many source features are redundant with many created features produces more complex and realistic redundancies; and
- whether these multivariate redundant features are more challenging for FS algorithms than the univariate ones created by the first approach.

5.1.2 Chapter Organisation

Sections 5.2 to 5.5 address the first three goals of this chapter by introducing the first approach (GPRFC) to automatically generating r.fs

for benchmarking FS algorithms, using GP. The quality of this approach is evaluated by applying common machine learning and FS algorithms to datasets with and without r.fs added to them, and examining how the results change. Sections 5.6 to 5.8 tackle the fourth, and fifth goals, by extending GPRFC to create r.fs that exhibit multivariate redundancy relationships, in an effort to make the r.fs even more realistic and challenging for FS algorithms. The performance of this extended GPMVRFC approach is compared to GPRFC across a variety of FS algorithms. A summary of the key findings of this chapter is provided in Section 5.9.

5.2 Creating Univariate Redundant Features: GPRFC

This section details the proposed method for automatically generating (univariate) redundant features (r.fs), including the GP representation, fitness function, and other important considerations made when designing the method. This method is named Genetic Programming for Redundant Feature Creation (GPRFC). The overall design of GPRFC is shown in Figure 5.1.

5.2.1 GP Representation

In this work we use a multi-tree GP representation, where each GP individual contains n distinct trees rather than a single tree. Each tree in an individual represents a single mapping (function) from the source feature (X), to a new redundant feature (Y). Using a multi-tree representation allows us to generate multiple r.fs per source feature, while encouraging each r.f to be distinct (less redundant) from all other r.fs. By generating a variety of r.fs, we increase the diversity of the types of redundancies between the source and redundant features. For example, an r.f Y_1 may have a polynomial relationship with X, whereas a second r.f Y_2 could have an exponential or trigonometric relationship —

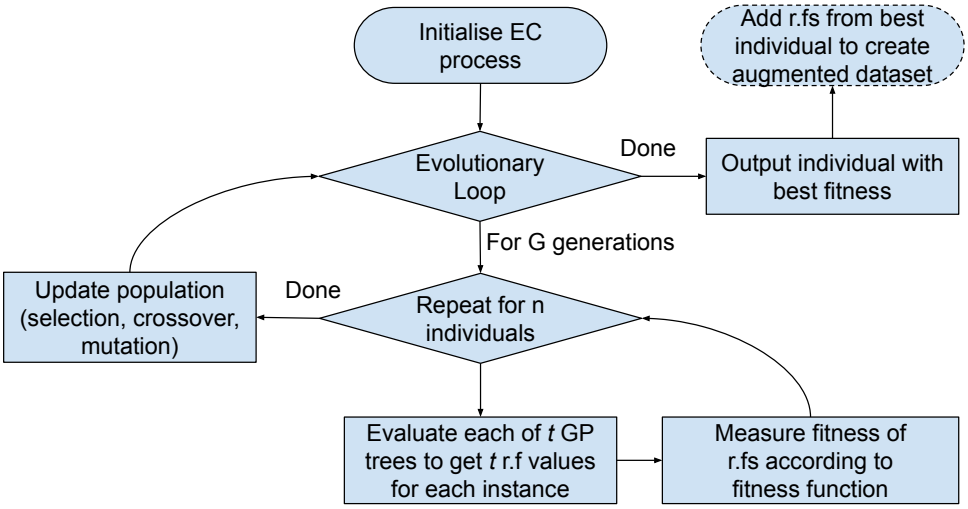


Figure 5.1: The overall flow of the GPRFC method. The dotted box represents the final augmented dataset that is the output of the redundant feature creation process.

both Y_1 and Y_2 are highly redundant with X, but less redundant with each other. This behaviour is encouraged by the fitness function, which will be discussed in more detail in Section 5.2.3. As each individual uses only one source feature to create r.fs, it is necessary to perform one run of the GP evolutionary process for every feature that redundant features need to be created for. However, each of these runs is expected to be relatively fast, as each instance effectively has only $1 + n$ features values used in the fitness function computation. Furthermore, these runs are fully independent, and so can be performed in an embarrassingly parallel manner.

5.2.2 Function and Terminal Sets

We use only a single terminal in this work: the source feature, X. We purposefully do not use a random value input (unlike most GP methods), as such a value is unlikely to meaningfully increase MI, and increases the search space unnecessarily.

In designing the function and terminal sets, it is important to have a wide range of operators with distinct behaviours, so that a variety of

redundancy relationships can be constructed in different trees. Based on this, we use a range of different arithmetic, trigonometric, and conditional operators as follows:

- Unary operators (taking one input): $\sin(a)$, $\tan(a)$, $\tanh(a)$, $\log(a)$, e^a, $\sqrt{a}$, a^2, a^3, and $-a$. We purposefully exclude $cos(a)$ due to its similarity to $sin(a)$. While a^2 and a^3 can be easily constructed in a GP tree, we include them as useful "building blocks".
- Binary operators (with two inputs): $a + b$, $a \times b$, $\max(a, b)$, $\min(a, b)$, and a^b. We exclude $a - b$ and $a \div b$ as they are the complements of addition and multiplication, and as they were found to negatively affect the learning process by easily producing constant values (i.e. $X - X = 0$, $X \div X = 1$).
- A single ternary operator, if, which outputs the second input if the first input is non-negative and the third input otherwise. This operator, in addition to $\max$ and $\min$, allows complex conditional behaviour and non-continuous functions to be generated.

5.2.3 Fitness Function

Our proposed fitness function is based on the concept of MI, a measure of the dependency between two features. We use MI as a proxy to measure the redundancy of a generated feature: if the MI between the source and generated feature is high, the generated feature is said to be highly redundant. Hence, the MI between the source and each generated feature/tree should be **maximised**. In addition, we choose to **minimise** the MI between each pair of generated features. In doing so, we implicitly encourage a set of r.fs that are redundant in *different ways* to be generated — for example, if two r.fs both had linear redundancies with the source feature, they would also have a high MI between them. This decision automatically increases the complexity of the generated r.fs, which should also make them harder for FS algorithms to remove. We describe the formulation of the fitness function in detail below.

Let X be the source feature, I be the GP individual whose fitness is being measured, which contains a set of trees (T), where n is the number of trees. Let the "baseline" MI, Ψ (used as a normalisation factor), be defined as the output of the MI estimation algorithm for $\Psi = MI(X, X)$[1]. In measuring the quality of I, we consider the **minimum** MI between any X and any r.f (called minSourceMI), as well as the **maximum** mean MI between any r.f and all other r.fs (called maxSharedMI). The quality of I is measured by how much more redundant the r.fs are with X than with each other, defined as follows:

$$\text{minSourceMI} = \min_{t \in T} \frac{MI(X, t)}{\Psi} \tag{5.1}$$

$$\text{maxSharedMI} = \max_{t \in T} \frac{\sum_{y \in T, y \neq t} \frac{MI(t,y)}{\Psi}}{n - 1} \tag{5.2}$$

$$\text{Quality}_I = \text{minSourceMI} - \text{maxSharedMI} \tag{5.3}$$

While this quality measure is expected to be suitable as a fitness function, it does not consider that having a minSourceMI below a certain threshold means that the r.fs produced are in fact not very redundant at all. In addition, generally a lower minSourceMI leads to a higher potential fitness, making the fitness function biased towards creating a set of r.fs that are very unrelated to each other, and only weakly related to the source feature. To remedy this, we introduce an additional component to the fitness function for when the minSourceMI is below some threshold, Θ, where Θ is the minimum "acceptable" redundancy between a r.f and X. In other words, individuals not meeting this criterion can be thought of as *infeasible solutions*. For these infeasible solutions, we do not consider the shared MI between r.fs to be important, as at least one of the r.fs is not acceptable. To encourage increasing the redundancy of each r.f in this scenario (i.e. encouraging the solution towards becoming feasible), we

[1]While $MI(X, X)$ is defined to be 1, an estimation algorithm must be used when MI is continuous: these algorithms do not guarantee their output has an upper limit of 1, and so here we normalise by the output of the estimator instead. This ensures that the normalised MI is at most 1, which is important in our fitness function's formulation.

penalise individuals based on the mean MI between the source and each r.f:

$$\text{Penalty}_I = \frac{-1}{\text{meanSourceMI}} \tag{5.4}$$

$$\text{meanSourceMI} = \frac{\sum_{t \in T} \frac{MI(X,t)}{\Psi}}{n} \tag{5.5}$$

This penalty function is designed as such so that the higher the meanSourceMI, the lower the penalty applied. Our fitness function is then the combination of these two functions:

$$\text{Fitness}_I = \begin{cases} \text{Quality}_I, & \text{if minSourceMI} \geq \Theta \\ \text{Penalty}_I, & \text{otherwise} \end{cases} \tag{5.6}$$

As the Penalty term of the fitness function is constrained to be less than 0, an individual with minSourceMI $\geq \Theta$ will nearly always be better than one that does not meet the Θ threshold. As our measurements of MI are normalised by Ψ, the threshold Θ can be chosen (roughly) from the range $[0, 1]$, where a value of $\Theta = 0$ corresponds to all r.fs being independent to X, and a value of $\Theta = 1$ corresponding to all r.fs being perfectly redundant with X. In practice, we found a Θ in the range $[0.6, 0.7]$ was a good choice for $n = 5$.

5.2.4 Further Considerations

A number of other factors needed to be addressed in order to achieve good results with the proposed method. These are discussed in turn below.

To improve the consistency of the GP method, the source feature was scaled so that all values fall in the range $[0, 1]$. However, this meant that at least one source feature value would be exactly 0. An input of 0 to the GP tree was found to significantly affect training as it would often result in multiplication or division by 0 within the tree. The common occurrence of dividing by 0 was particularly troublesome, as it meant the

tree would not produce a valid output, making the whole individual invalid. To remedy this, we added a small weighting to each feature value, of size ϵ, such that all feature values lie in $[0 + \epsilon, 1 + \epsilon]$. In this work, we found setting $\epsilon = 1 \times 10^{-3}$ to be suitable.

While the above scaling approach is expected to work well on artificially-generated datasets, it does not address an issue with many real-world classification datasets: duplicate feature values. Consider the example of a (real-world) dataset where a feature takes values in $\{1, 2, 3, 4\}$. Given there are only four unique inputs to a GP tree, the tree may only produce (at most) four unique outputs. This greatly limits the ability of GP to learn to create multiple distinct r.fs as only very "coarse" r.fs can be generated (with low complexity). To address this performance limitation, we add a small amount of stochastic noise (using a constant seed) to each source feature value, so that each feature value is likely to be distinct. This is essentially equivalent to changing the input of the GP tree to be $X + \delta$, where δ is a small value that is constant for a given value of X. As before, we ensure δ is strictly positive. The feature values are hence in the range $[0 + \delta, 1 + \delta]$, where we defined δ to be a random number between 0.001ϵ and ϵ. In both the above approaches, we still evaluate the MI between a r.f and X (i.e. when computing the fitness function) using the **original** (i.e. unscaled) feature values, to ensure we measure the true redundancy.

In addition to scaling the source feature, we also scale the constructed redundant features to lie in $[0, 1]$. This serves two purposes: it ensures the r.fs have "sensible" ranges, and so can be more easily visualised, and it also means they have the same range as the source feature, which is important for many algorithms such as k-nearest neighbour, k-means clustering etc. Finally, the redundant features are rounded to 5 decimal places, to prevent GP from evolving very sensitive features whose precision may be lost when saved to file or used in another algorithm.

Other Parameter Settings:

We use a relatively high max tree depth of 15 and mutation rate of 40% (with crossover of 60%). Using a high max tree depth was found to encourage more complex trees to be formed, which tended to produce more complex features. Evaluating the larger trees is not significantly more costly, as the computation of MI is the most expensive part of the fitness evaluation. 40% mutation was used to encourage the generation of more diverse trees — however, crossover is still important to ensure that useful function "building blocks" are passed between different GP individuals. The population size was set to 1,024, and top-10 elitism was used, as standard. In this work, we used $n = 5$ trees as it was found to produce a reasonable balance between making a large number of r.fs and making highly diverse r.fs. Decreasing n will produce r.fs that are less redundant to each other, whereas increasing n will give more, but less distinct r.fs. Θ was set to 0.7 in this work based on preliminary tests.

5.3 Experiment Design

We tested the proposed GPRFC approach on a number of popular datasets, as listed in Table 5.1. These datasets include three classification datasets from the UCI repository [31], two of which are quite simple and easy to classify well (Iris and Wine), whereas the third (Vehicle) is more challenging as it contains more features, instances, and classes. We also use two synthetic clustering datasets (10d10c and 10d40c), which have 10 and 40 clusters respectively and are generated using an Ellipsoidal cluster generator [56]. The datasets chosen all have a small number of features to reduce the number of GP runs required.

For each dataset, five r.fs are created per source feature, to give a result of $d+5d = 6d$ features for d source features. As the feature creation approach uses GP, it is stochastic, and so at least 30 runs were performed on each dataset.

To evaluate the created r.fs, we used the classifiers, clusterers, and feature

Table 5.1: Datasets used in the experiments.

Name	No. Features	No. Instances	No. Classes/Clusters
Iris	4	150	3
Wine	13	178	3
Vehicle	18	846	4
10d10c	10	2903	10
10d40c	10	2023	40

selection algorithms provided by the WEKA [55] package. We selected four varied and popular classifiers: the J48 Decision Tree (DT) algorithm, k-nearest neighbour (KNN, with $k = 3$), Naive Bayes (NB), and the Sequential Minimal Optimisation implementation of the Support Vector Machine (SVM). For clustering, we use three different varieties of clustering algorithms: k-means++, agglomerative clustering (the average-link variant), and the Expectation Maximisation (EM) algorithm.

5.4 Results and Discussion

As there are no known redundant feature creation methods that use a guided search to automatically find good r.fs, we are unable to directly compare GPRFC to a known baseline. Instead, we directly evaluate the quality of the r.fs created across the datasets in terms of the fitness achieved. If GPRFC is able to achieve a high fitness on a given dataset, then it indicates it has successfully created a set of diverse features that have high redundancy with the source feature, but low redundancy with each other.

We also investigate how the addition of the r.fs affects the performance of some common classification and clustering algorithms, and how well some simple FS algorithms are able to identify (and remove) the added r.fs, in order to evaluate the suitability of the proposed method for creating benchmark datasets. If the clustering/classification performance drops, or simple FS algorithms struggle to remove added r.fs, this would indicate that GPRFC has the potential to produce hard-to-detect, misleading features that are suitable for benchmarking FS algorithms.

Table 5.2: Fitness achieved by GPRFC across all features on each dataset. Standard deviation is taken across the means for each feature. 30 runs were performed per feature per dataset.

Dataset	Mean	Std. Dev
Iris	0.333	0.082
Wine	0.203	0.055
Vehicle	0.351	0.041
10d10c	0.106	0.010
10d40c	0.141	0.006

5.4.1 Fitness

Table 5.2 shows the performance of GPRFC in terms of the average fitness achieved across the tested datasets. GPRFC achieves a high fitness on two of the three classification datasets: Iris and Vehicle. A mean fitness of 0.351 on Vehicle indicates that the typical created r.f is 35.1% *more redundant* with the source feature than the other created r.fs, for example, 75.1% MI with the source feature vs only 40% MI with the other created r.fs. The performance on the two synthetic clustering datasets is not as strong, but the created r.fs are still clearly more redundant with the source feature than each other.

In general, it appears that datasets containing fewer instances tend to have a higher standard deviation — perhaps as the fitness is more sensitive to any one single feature value being altered during the evolutionary process. The fitness across the Iris dataset, which has the highest standard deviation, is shown in Table 5.3 for each feature. This table clearly shows that F2 has a much lower mean fitness than the other features, and so gives a high standard deviation on the Iris dataset. It is not obvious as to why GPRFC can learn more effectively on certain features. One explanation may be that as GPRFC produces functions that transform the feature space, features that have very dense feature value distributions are harder to transform with a high level of granularity, and so harder to optimise. However, further investigation is needed.

Table 5.3: Fitness achieved by GPRFC across the 30 runs on the Iris dataset.

Feature	Mean	Std. Dev
F0	0.362	0.050
F1	0.398	0.036
F2	0.213	0.029
F3	0.359	0.053

5.4.2 Classification Performance

The performance of a number of classifiers on the original datasets compared to the datasets with added r.fs ("augmented datasets") are shown in Table 5.4. In general, performance is very consistent between the original and augmented datasets — in most cases, dropping by 2-3%, or holding steady. Given that redundant features are not inherently misleading to a classifier, it makes sense that performance may not drop much – though the classification model produced will certainly be more complex. Two major exceptions to this are on the KNN classifier, which had a decrease of around 5% and 11% accuracy on Iris and Vehicle respectively. This is likely due to the created r.fs not having the same distances between instances' feature values as the source features had. As KNN is a distance-based classifier, any addition of features that transform the feature space non-linearly will directly alter the distances between instances. Testing on more difficult datasets with many more features may show a bigger decrease in performance, as the search space may become complex/large enough to better challenge classification algorithms. The small increase in performance on the Vehicle dataset with NB is not statistically significant.

5.4.3 Clustering Performance

The performance of three clustering algorithms on the original and augmented datasets was investigated, with the results shown in Table 5.5. As with the classification datasets, there is generally little change in performance — in fact, performance appears to slightly

Table 5.4: Test classification accuracy on each of the datasets before ("Original") and after ("Augmented") the created r.fs were added. Each of the 30 runs of GPRFC produced one augmented dataset — hence, the mean and standard deviation accuracy on these 30 augmented datasets are reported. A split of 70% training to 30% test was used.

Method	Iris		Wine		Vehicle	
	Original	Augmented	Original	Augmented	Original	Augmented
DT	0.978	0.956±0.004	0.981	0.977±0.017	0.709	0.692±0.025
KNN	1.000	0.947±0.035	0.962	0.961±0.028	0.720	0.613±0.029
NB	0.978	0.964±0.018	1.000	0.979±0.018	0.465	0.490±0.025
SVM	0.978	0.968±0.020	0.981	0.974±0.016	0.740	0.715±0.018

Table 5.5: Adjusted Rand Index of the clusters produced on each of the datasets before ("Original") and after ("Augmented") the created r.fs were added. Each of the 30 runs of GPRFC produced one augmented dataset — hence, the mean and standard deviation accuracy on these 30 augmented datasets are reported. k-means++ and EM are stochastic algorithms and so the mean of 30 runs per augmented dataset was used.

Method	10d10c		10d40c	
	Original	Augmented	Original	Augmented
k-means++	0.548	0.558±0.023	0.445	0.491±0.019
Agglomerative	0.495	0.528±0.064	0.276	0.309±0.046
EM	0.588	0.606±0.014	0.433	0.520±0.011

increase when adding the created r.fs. However, the clusters produced are more complex and less interpretable — with $6d$ features per instance compared to only d in the original datasets. The main exception to this pattern is the EM method on the 10d40c dataset, where the addition of redundant features increases the clustering performance. This is unexpected, but may suggest that the EM algorithm is likely to create a greater number of clusters when a larger feature space is available (even though it is redundant in nature).

5.4.4 Feature Selection Results

Feature Ranking:

A common technique used in supervised feature selection is to measure how well a given feature can be used to predict the class label for a set of instances. Information Gain (IG) [72] is often used as a metric to measure this, using similar principles to MI. To see how "confusing" our created r.fs may be to a FS algorithm, we ranked the features of the median and best results of applying GPRFC to the Iris dataset, using IG as shown in Table 5.6. We use the Iris dataset as our example as it has the fewest features, and so can be analysed most easily.

The majority of created r.fs have similar rankings to their source features, with the top half of the ranks taken by F2 and F3, and the bottom half by F0 and F1. This is unsurprising — given that the created r.fs share a high amount of information with the source features, they are likely to also have a similar ability to predict the class label. However, the r.fs do have small variances in their IG value compared to their source features: for example, on the median result, F2 has an IG of 1.418, and its r.fs have IG values between 0.864 and 1.367. On the best result, F2c and F2e actually have **better** IG than the source feature; F2's r.fs range in IG value from 0.827 to 1.456. These results indicate that while the created r.fs clearly share information with their source features, they are still different enough that their redundancy is non-trivial to identify and they are likely to have an effect on the classification task.

Using a FS algorithm:

To further investigate how suitable the created r.fs are for benchmarking FS algorithms, we applied a basic FS algorithm to the same two augmented Iris datasets. We used the canonical Sequential Floating Forward Selection (SFFS) [138], which is an extension to the Sequential Forward Selection (SFS) algorithm. SFS starts with no features selected, and iteratively adds the best of the remaining unselected features, until

Table 5.6: Features ranked by Information Gain (with respect to the class label) on the augmented datasets created by the median (5.6a) and best (5.6b) runs of GPRFC.

(a) Median.

Info Gain	Feature
1.418	F2
1.385	F3a
1.378	F3
1.367	F2b
1.274	F3c
1.216	F2a
1.163	F3e
0.976	F3d
0.918	F2d
0.908	F3b
0.864	F2e
0.705	F0a
0.698	F0
0.554	F2c
0.376	F1e
0.376	F1b
0.376	F1
0.364	F0b
0.325	F1d
0.177	F0c
0.118	F0d
0.098	F0e
0.000	F1a
0.000	F1c

(b) Best.

Info Gain	Feature
1.456	F2c
1.421	F2e
1.418	F2
1.378	F3
1.313	F3a
1.295	F2b
1.214	F3d
1.098	F3c
1.077	F3e
1.057	F3b
0.918	F2d
0.827	F2a
0.722	F0a
0.698	F0
0.597	F0e
0.597	F0d
0.419	F0c
0.376	F1c
0.376	F1
0.198	F1d
0.158	F1b
0.089	F1a
0.000	F0b
0.000	F1e

performance is not improved by adding the next feature. SFFS follows the same procedure, but also performs a backward search after each addition of a new feature. That is, it repetitively removes the worst feature in the selected subset, until performance is not improved by removing an additional feature. This floating search helps to prevent the FS algorithm getting stuck in local optima, and makes SFFS one of the most commonly used deterministic FS algorithms.

In this work, we used a wrapper method where the SVM algorithm is used to classify the dataset for a given feature subset, and the accuracy of

the results is used as the performance of the selected features. We use the SVM classifier as it had the highest performance in Table 5.4. Our SFFS implementation used a training set to train the SVM, and a "validation set" [2] to evaluate the performance of the SVM on unseen data during the training process. This "validation set" gives the expected performance of an SVM using the selected features on future unseen data, and is used to measure the quality of the selected features. Finally, we use a separate unseen test set to measure the quality of the final selected features on unseen data. The training, validation, and test sets are 60%, 20% and 20% of the shuffled dataset, respectively.

On the median dataset (for Iris), this FS method selects features [F2b,F3] with a test accuracy rate of 0.933. On the best dataset, [F0,F1a,F3] are selected with an accuracy of 0.967. On the original dataset, the FS method selects only F3, with an accuracy of 0.967. While the obtained classification accuracy on the augmented datasets is similar, the FS method clearly selects extraneous features, which gives a more complex model than that of when only a single feature is selected on the original dataset.

Given that obtaining good performance on Iris is easy, and so FS is also relatively easy, we performed a similar experiment on the Vehicle dataset to see if different behaviour occurs on a harder, higher-dimensional problem. On the original 18-dimensional vehicle dataset, the SFFS method (as described above) selects 12 features: [F0, F2, F5, F7, F8, F9, F10, F12, F13, F16, F17], with a test accuracy of 0.710. On the median augmented dataset however, it selects seven features: [F3, F9b, F12d, F12e, F13, F17, F17e] with a test accuracy of only 0.473. The FS method has clearly struggled to find a good set of features, as it selects multiple redundant features while also failing to select many features that were selected in the original dataset. Furthermore, the training accuracy was

[2] Really, it is a test set, which is applied to an SVM trained with the training set, but we refrain from using this term to distinguish from the true test set, which is only ever used at the conclusion of the search to measure the quality of the features found by the search process as a whole.

reasonably similar for both datasets (0.769 and 0.686 for the original and augmented respectively), indicating that the created r.fs were able to mislead the FS algorithm well enough to prevent a well-generalised classifier from being produced. Further investigation is needed to provide more quantitative evidence that GPRFC produces r.fs that make difficult benchmark datasets, but the preliminary results are a promising sign that the proposed method has potential.

5.5 Further Analysis

While we have shown that GPRFC is able to automatically produce a set of redundant features that have high MI with an original feature, it is not yet obvious **how** it is able to do so. To investigate this aspect, we plotted the created features against the original features for each of the four features on the Iris dataset, using the median result of the 30 runs of GPRFC. We choose to analyse Iris as it is the dataset with the smallest feature set. These plots are shown in Figure 5.2.

The most striking observation of these plots is that the functions produced by GPRFC are incredibly varied — in fact, nearly every plot has a distinct appearance. The functions are also clearly complex, with no linear relationships apparent. A few functions are somewhat recognisable: for example, F2a (similar to a sine wave), F2b (a power curve), and F2d (a polynomial). The function evolved for F0a is similar in appearance to a sigmoid function, despite the sigmoid not being directly in the function set. Many of the remaining functions are more difficult to classify, as they either appear to have a number of different components (e.g. F0b, F3b), or have a majority of instances at a similar scale (e.g. F0e, F1b, F3a, F3c).

While generally each set of r.fs for a given source feature appear to be quite distinct, F3a and F3c appear to be very similar. This behaviour is counter-intuitive, as the fitness function directly penalises a r.f being similar to other r.fs for the same source feature. Indeed, F3a and F3c have

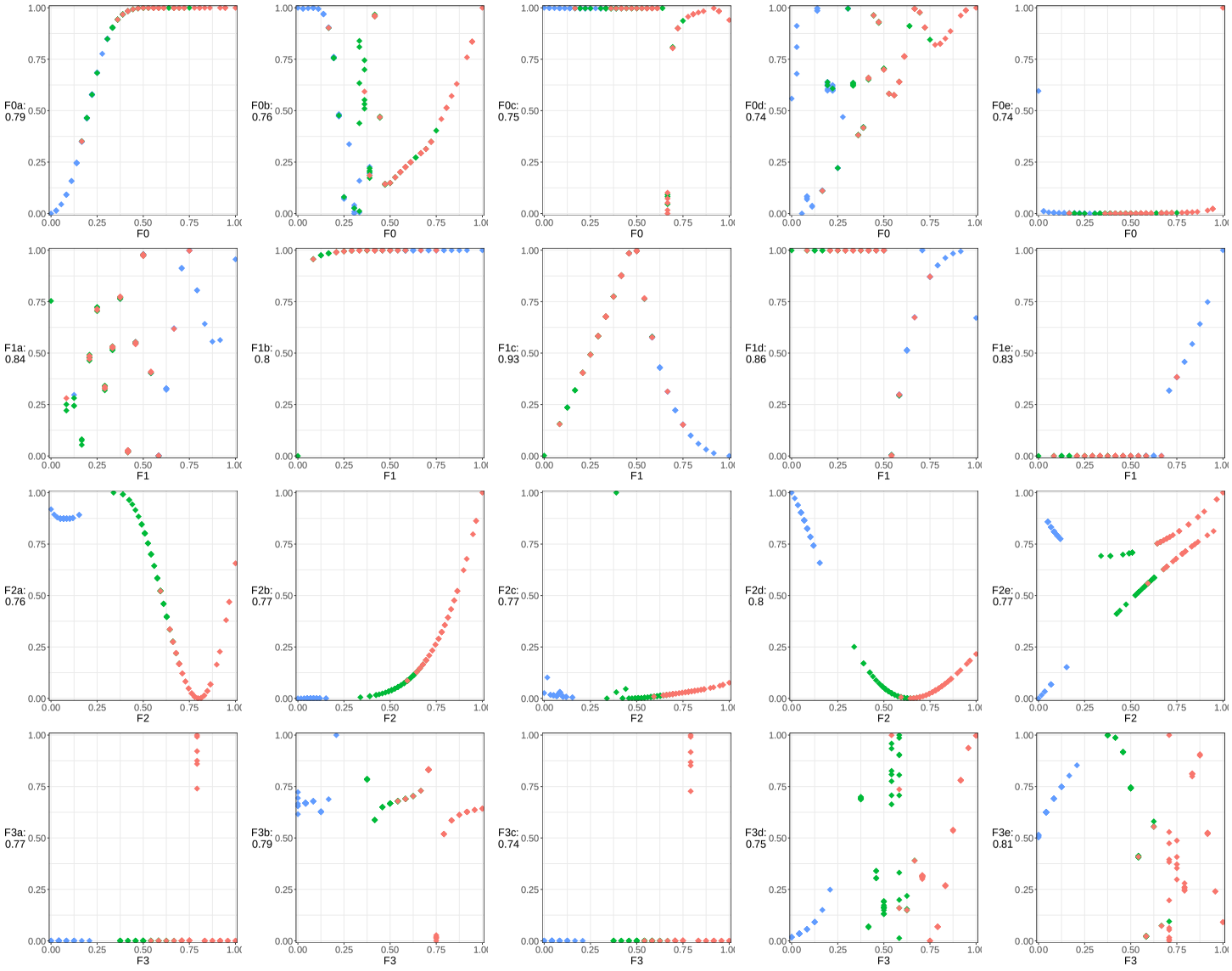


Figure 5.2: Source feature (x-axis) plotted against the five r.fs (y-axis) for each of F0 (1st row) to F3 (4th row) on Iris. The y-axis title is the name of each r.f and its source MI. Points are coloured red, green, or blue to indicate they belong to the setosa, versicolor, or virginica classes respectively. A small amount of jitter is added to each point to distinguish overlapping points.

```
F3a = (pow (+ (exp (sqrt
(cube X)))
(neg (log X))) (tan (exp (+ (log X) (mul X X)))))

F3c = (pow (+ (exp (sqrt
(sqrt (tan (square (square (max (sin (exp X)) (sin X))))))))
(neg (log X))) (tan (exp (+ (log X) (mul X X)))))
```

Figure 5.3: Example trees produced by GPRFC for F3a and F3c. The entire first and third lines are shared by both trees.

a MI of 0.83 — however, they each have very low MI (a maximum of 0.14) with the other r.fs (F3b/d/e), which means their average shared MI is still very low, at 0.34. The two trees corresponding to these two features are very similar (see Figure 5.3). This issue may be alleviated by adapting the fitness function to consider the **worst-case**: that is, what is the highest value a given r.f shares with another r.f?

5.6 Creating Multivariate Redundant Features: GPMVRFC

GPRFC was the first GP-based approach to automatically create redundant features from existing data for benchmarking FS algorithms. While GPRFC was able to produce significantly more complex (non-linear) feature redundancies than existing (naïve) approaches, these redundancies were univariate. In many complex real-world datasets, the redundancy relationship between features is more realistically a multivariate one — many features may be jointly redundant with many other features. In other words, many-to-many redundancies are more common than simpler one-to-many relationships. Detecting multivariate redundancies is also substantially more difficult for FS algorithms, as the search space is strictly larger, given the exponential number of possible redundancies in a many-to-many situation. Clearly, in order to produce

benchmark datasets that can challenge the most powerful FS algorithms in a realistic manner, there is a need for methods that can create multivariately redundant features. The remainder of this chapter proposes an extension to GPRFC that produces multivariate redundant features: Genetic Programming for Multivariate Redundant Feature Creation (GPMVRFC). The overall flow of GPMVRFC is the same as GPRFC (see Figure 5.1, page 152), but with key changes to the program structure and fitness function used.

5.6.1 GP Representation

The proposed GPMVRFC method follows the general structure of that proposed in GPRFC. Each GP individual has n trees, each representing a functional mapping from some number of source features ($\mathbf{X}$) to a single created r.f (Y). The number of source features used is dependent on the size of the original feature set; the bigger the original feature set, the more source features that can be used to create a single set of r.fs. In this work, we propose using the following equation:

$$|\mathbf{X}| = \lfloor \max\left(2, \min\left(\frac{m}{4}, 5\right)\right) \rfloor \tag{5.7}$$

where m is the size of the original feature set, and $|\mathbf{X}|$ the number of source features. Computing the number of source features in this manner ensures that at minimum two source features are used, at least four *sets* of r.fs are created (for $m > 8$), and that at most five source features are used (so as to keep complexity in scope). Note that for all $m \geq 20$, five source features will be used. $|\mathbf{X}|$ features of the dataset will be used in a single GP run, as the terminal set, and so $\frac{m}{|\mathbf{X}|}$ runs must be performed to create the full augmented dataset. In contrast, GPRFC required m runs — hence, GPMVRFC requires only 20% of the runs given $m \geq 20$. Note that although each tree has five source features (terminals) to draw from, there is no requirement that every tree uses all the source features, and hence different trees may be redundant with different subsets of the source features.

Table 5.7: Function set used in GPMVRFC.

Operator Type	Function	#Inputs
Arithmetic	$\sqrt{a}$	1
	a^2	1
	a^3	1
	$-a$	1
	e^a	1
	$\log(a)$	1
	$a + b$	2
	$a \times b$	2
	a^b	2
Trigonometric	$\sin(a)$	1
	$\tan(a)$	1
	$\tanh(a)$	1
Conditional	$\min(a, b)$	2
	$\max(a, b)$	2
	if(a, b, c)	3

5.6.2 Function and Terminal Sets

The terminal set consists of only the $|\mathbf{X}|$ source features of the dataset that are being used to create r.fs of that specific GP run. No random value terminal is used as it would not add any useful component to the mapping function. The function set is unchanged from GPRFC (as this is not the focus of this work), and is summarised in Table 5.7.

5.6.3 Fitness Function

One of the most difficult aspects of this task is producing created features that represent different *types* of redundancies between the source and created feature/s. For example, creating many redundant features that all have a strong polynomial redundancy with a source feature would maximise MI very well, but does not represent a challenging/diverse benchmark for FS algorithms. GPRFC used a fitness function that minimised the MI between created features, which does not directly compare how different the types of redundancies the created features share with the source feature are. In addition, such a fitness function is

expected to perform poorly in the multivariate case, as it is necessary for the created features to be redundant with each other in order to be jointly redundant with the source features. Finally, the fitness function itself was expensive due to the number of MI calculations required, and as a single GP run was needed for every feature in the original dataset. To address these issues, GPMVRFC uses a fitness function with a different approach.

The fitness function used in this work consists of two components: one that measures the redundancy between the source feature set and the created feature set using multivariate MI, and another that measures the difference between the created features by comparing the gradients of the distributions of each created feature. The fitness of an individual is based on the source features being used ($\mathbf{X}$), and the features created by that individual ($\mathbf{Y}$), i.e. the outputs of all the GP trees once they have been evaluated using each instance in the dataset. Measuring the redundancy between the source and created feature sets is particularly straightforward (and efficient) in a multivariate scenario:

$$\text{Redundancy} = MI(\mathbf{X}, \mathbf{Y}) \text{ [as per Equation (2.23), page 42]} \tag{5.8}$$

As we wish to create features that are as redundant as possible (while being redundant in different ways), we maximise the MI between $\mathbf{X}$ and $\mathbf{Y}$. As discussed previously, an estimator is used to compute the approximate MI as the underlying *pdfs* of $\mathbf{X}$ and $\mathbf{Y}$ are not known.

To measure the similarity of the created features, we designed a novel approach based on the distribution of each created feature compared to the source features. The intuition behind this approach is that two created features should have significantly different distributions across the source feature space if they are to be considered to have different "types" of redundancy with the source features. Figure 5.4 shows an example of some potential simple created features that GPMVRFC could create, where the y-axis represents the output of a GP tree given some input feature. The linear, sine, and polynomial functions clearly have

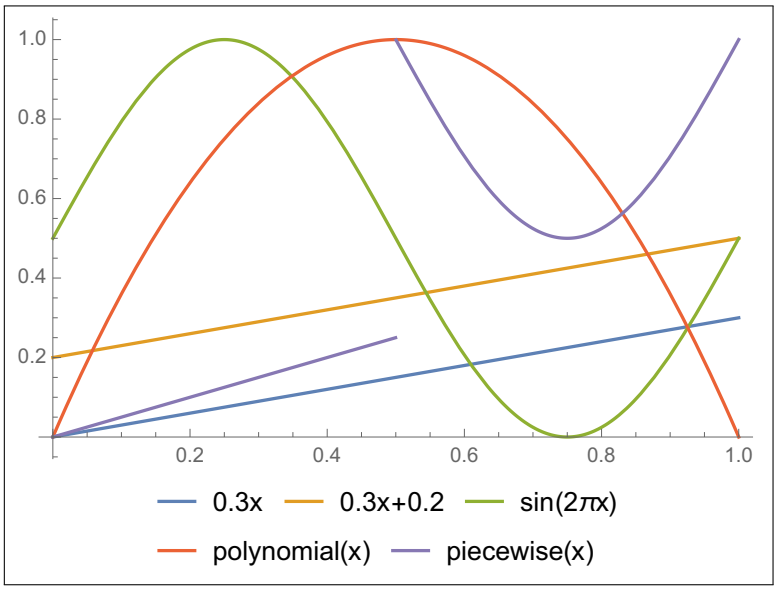


Figure 5.4: Example of some (scaled) redundancy mappings that could be created by GPMVRFC.

different shapes in the feature space, and so represent different types of r.fs. The two linear functions, however, both are the exact same type of r.fs, except that they have different offsets from the x-axis. The piecewise and sine functions have the same redundancy function for half of the feature space but are very different in the other half. Based on this observation, we propose comparing the **gradients** of each created feature for each value in the source feature space, in order to evaluate the "semantic" difference between the created features. Note that when considering the gradient, the two linear functions are identical, and the piecewise and polynomial functions have identical gradients for half the feature space. This approach is complicated slightly in that there are multiple source features that can be used to produce different distributions of the created r.fs — we propose an algorithm to cope with this below.

For each created feature in an individual $\mathbf{A} \in \mathbf{Y}$, we calculate the difference between $\mathbf{A}$ and each other created feature $\mathbf{B} \in \{\mathbf{Y} - \mathbf{A}\}$ using the approach shown in Algorithm 3. We take the minimum of these

Algorithm 3: Computing the difference of two created feature vectors, **A** and **B**, given source features **X**.

Difference$_{min}$ = minimum of *ComputeDifference*$(\mathbf{A}, \mathbf{B}, x)$ for each $x \in X$
return *Difference*$_{min}$
Function *ComputeDifference*(A, B, x)**:**
    $A' = Sort(\mathbf{A})$ according to the natural ordering of x.
    $B' = Sort(\mathbf{B})$ according to the natural ordering of x.
    $Gradient_{diff} = 0$
    for $i \in [1, |x|)$ **do**
        $Gradient_{Ai} = A'[i] - A'[i-1]$
        $Gradient_{Bi} = B'[i] - B'[i-1]$
        $Gradient_{diff} += |Gradient_{Ai} - Gradient_{Bi}|$
    end
    return $\frac{1}{|x|} \times Gradient_{diff}$

differences as the minimum difference between **A** and any other created feature, called RFDiff$_{\text{min}\mathbf{A}}$. In addition, we also consider the difference between **A** and each of the source features (using Algorithm 3), to ensure that no created feature inadvertently reconstructs one of the existing features (i.e. making it naïvely redundant). The minimum difference between **A** and any source feature $x \in X$ is computed as SourceDiff$_{\text{min}\mathbf{A}}$. The smaller of RFDiff$_{\text{min}\mathbf{A}}$ and SourceDiff$_{\text{min}\mathbf{A}}$ is the distance between **A** and any other constructed feature or source feature. The final formulation of the difference of the created r.fs is as follows:

$$\text{Difference}_{\text{RFs}} = \min_{\mathbf{A} \in Y} \min\{\text{RFDiff}_{\text{min}\mathbf{A}}, \ \text{SourceDiff}_{\text{min}\mathbf{A}}\} \tag{5.9}$$

where the difference should be maximised to encourage diverse types of r.fs. The fitness of a given individual is thus as follows:

$$\text{Fitness} = \text{Redundancy} \times \text{Difference}_{RFs} \tag{5.10}$$

In the rare case where one or more trees represents an invalid solution (e.g. $\tan(0.5\pi) = \infty$), the fitness of the individual is instead set to negative the number of invalid trees. This fitness function is also significantly more efficient than the one proposed in GPRFC. GPRFC evaluated the MI of

every pair of created features, i.e. at a cost of $O(n^2)$. While our proposed fitness function compares every pair of created features also, the difference algorithm we use requires significantly less computational time than the MI estimator would otherwise use.

5.6.4 Other Details

As GPMVRFC uses a similar representation to GPRFC, we apply a number of additional "tricks" used in GPRFC to the proposed method also. These are as discussed in Section 5.2.4.

5.6.5 GP Parameters

We used a population size of 1,024, 200 generations of evolution, 40% mutation, 60% crossover, top-10 elitism and a max tree depth of 15. These parameter settings are similar to those in GPRFC in order to maintain a fair comparison. The one difference from GPRFC is that we employed $n = 10$ trees per individual — as multiple source features are being used, there are many more possible created features, and as we do not run the GP process once per source feature, more trees are needed per run in order to produce a reasonable number of r.fs.

5.7 Experiment Design: GPMVRFC

When evaluating GPRFC, we aimed to explore if GP had the *potential* to produce complex r.fs, and so we primarily looked at how well the created r.fs related to the existing features and how they affected classification and clustering performance. Our motivations with GPMVRFC were to make more challenging multivariate r.fs that are able to *mislead* FS algorithms, and so we focus on evaluating how the multivariate r.fs affect the FS algorithms' performance. Given that FS has been studied and used to a much greater extent in supervised learning, and to constrain the scope of our experiments, we do not use clustering datasets unlike when previously evaluating GPRFC. To evaluate the performance of the

Table 5.8: Feature set size of original dataset vs those augmented by GPMVRFC and GPRFC.

Dataset	Source #Features	GPMVRFC #Features	GPRFC #Features
Iris	4	24	28
Wine	13	52	78
WDBC	9	48	54
Dermatology	34	90	196
Vehicle	18	56	104
Image. Seg.	18	56	104
Movement Libras	90	270	540

proposed GPMVRFC method, we generated a number of augmented datasets by running GPMVRFC on a number of well-known and popular classification datasets from the UCI machine learning repository [31].

The number of source features and the total number of features[3] in the augmented datasets produced by GPMVRFC, and GPRFC, are shown in Table 5.8. As GP is a stochastic search method, we created 40 augmented datasets for each source dataset utilising different initial random seeds. These augmented datasets were then used to test a number of different FS methods, in order to evaluate how the addition of r.fs would affect the performance of FS methods in terms of the accuracy achieved, and the number of features selected. In order to provide a comprehensive evaluation, we use algorithms from all three main FS categories: filter, wrapper, and embedded methods:

Filter:

Ranking each feature in an augmented dataset according to its Information Gain (IG) [72, 102]. IG is used to evaluate how well a feature can predict the class label, and so if created features are redundant with source features, they may be likely to have a similar IG ranking. Different classifiers were then used to classify the dataset using the top n features

[3]That is, the sum of the number of source features and number of created redundant features.

(where n is varied from 1 to #features). Furthermore, we used two more advanced filter FS algorithms to further test the difficulty of performing FS on the augmented datasets. We used the L1-norm in a linear SVM ("l1SVC") [186] to perform FS (i.e. sparse feature selection), as well a Joint Mutual Information (JMI)-based search [180].

Wrapper:

Applying simple sequential search based FS algorithms to each augmented dataset, where a wrapped classifier was used to evaluate the selected subset at each stage. Sequential Forward Selection (SFS), Sequential Backward Selection (SBS), and the floating versions of SFS (SFFS) and SBS (SFBS) [160] were tested.

Embedded:

Finally, we also tested directly using the augmented datasets in a decision tree (DT) classifier. These tests used the scikit-learn library [129].

All of the stochastic FS methods listed above were run 30 times with independent seeds and averaged. The results of each of the above tests will be discussed in turn in the next section.

5.8 Results and Discussion: GPMVRFC

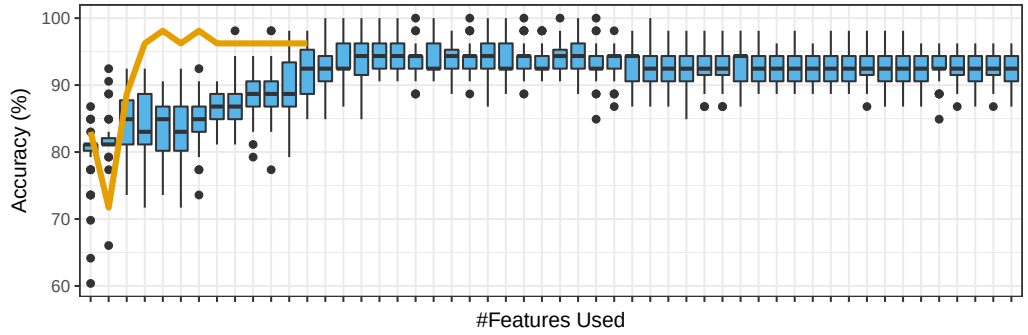
Due to the number of different test combinations across the seven datasets, we focus on analysing the results of the four datasets that show the clearest patterns and provide the most insight. We discuss each of the Wine, Dermatology, Vehicle and Image Segmentation datasets in turn in this section. For each dataset, we discuss the feature ranking performance (with the KNN classifier), the DT performance, SFFS and SFBS performance (again with KNN), and the l1SVC and JMI FS methods, using KNN as the classifier. KNN was chosen as it produced the most consistent patterns, and is a simple and efficient classifier. Our performance analysis considers both the number of selected features and

the test accuracy of the classifiers. For the feature ranking approach, we exclude the GPRFC method from the plots, as it is a univariate FS method and so is unlikely to give a fair comparison of the difficulty of the r.fs created by each method.

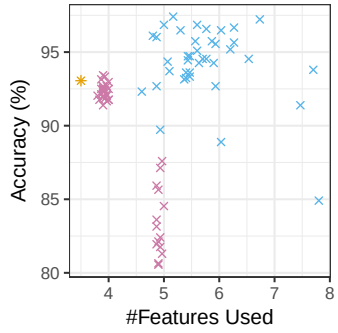
5.8.1 Wine

Figure 5.5 shows the results of the FS methods on each of the 40 augmented Wine datasets. Figure 5.5a shows how the accuracy of the KNN classifier varies when the top n ranked features are used, where n increases across the x-axis. The orange line represents the performance on the original Wine dataset using the same ranking process. We can clearly see that accuracy is lowered on the augmented datasets, due to the addition of r.fs that mislead the IG ranking process. However, at a certain point ($n \approx 15$), enough good features are selected for the accuracy to reach the same level that selecting four features on the original dataset would give.

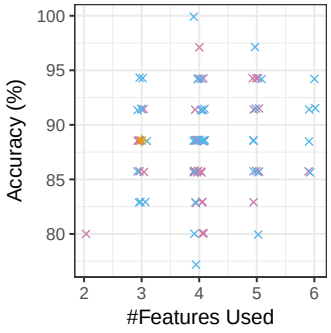
The remainder of the plots show one FS approach for each of the 40 augmented datasets for GPMVRFC and GPRFC, and the original dataset. A small amount of jitter is added so that duplicate points can be distinguished. The sequential FS algorithms (SFFS and SFBS; Figures 5.5c and 5.5d) show very similar accuracy distributions for each of the two methods. Given that GPMVRFC creates 26 fewer features, this suggests that the features created by GPMVRFC may be more difficult to identify, given that GPRFC has a larger FS search space. The l1SVC method (Figure 5.5e) appears to under-select features, especially on GPMVRFC, compared to on the original dataset, with a reduction in accuracy — suggesting that the method is being "confused". The JMI method (Figure 5.5f) is harder to interpret, though it appears that both methods cause extra features to be selected. The FS methods appear to struggle to select a lower percentage of features on the GPMVRFC datasets. The DT method (Figure 5.5b) is particularly interesting, as it shows GPMVRFC actually improving the performance of the classifier, albeit with more



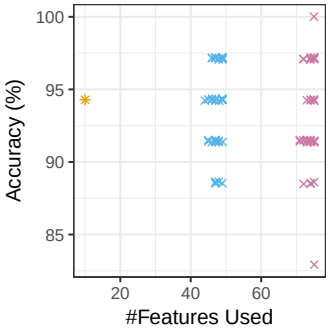
(a) Features ranked by IG, classified with KNN.



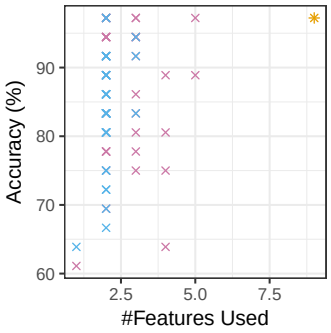
(b) Classified with DT.



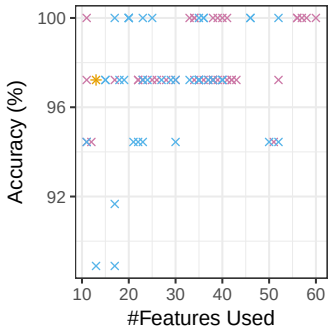
(c) FS with SFFS, classified with KNN.



(d) FS with SFBS, classified with KNN.



(e) FS with l1SVC, classified with KNN.



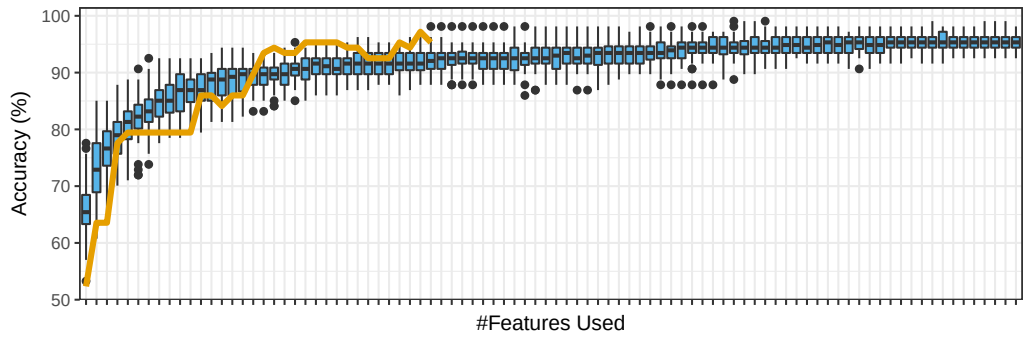
(f) FS with JMI, classified with KNN.

Figure 5.5: Wine Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is the existing GPRFC method.

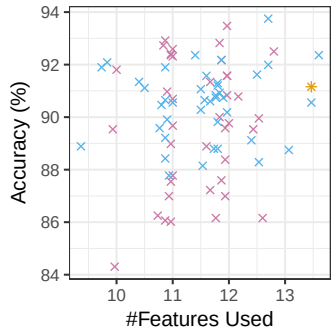
features being used. If the DT method purposefully selects additional features while improving accuracy, then GPMVRFC must be creating more powerful/better features than some of the original features. Considering that GPMVRFC combines several source features to create an r.f, it is not unexpected that some of these created r.fs may actually be of higher quality. Furthermore, real-world datasets such as those used here often have complex hidden feature interactions, which GPMVRFC may be able to "uncover" when it creates new r.fs. It may be that the multivariate features represent a sort of *kernel* that represents some implicit feature space that makes classification easier.

5.8.2 Dermatology

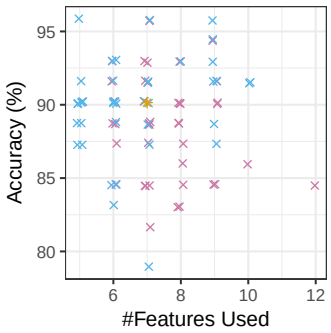
The results on the Dermatology dataset (Figure 5.6) further reinforces the hypothesis that GPMVRFC may actually be inadvertently creating more powerful features (for classification). Feature ranking (Figure 5.6a) shows that selecting the same number of features on the augmented datasets can give *better* test accuracy than on the original dataset for the first ≈ 15 features. This pattern continues on four of the remaining FS methods (Figures 5.6c to 5.6f) where GPMVRFC datasets generally have higher accuracy compared to GPRFC, and often even compared to the original dataset. In the case of SFFS, GPMVRFC often gives higher accuracy with fewer features. On l1SVC and JMI, using the same number of features as GPRFC gives higher accuracy. While this initially seems unsatisfactory, we note this may be partially due to fewer features being created by GPMVRFC than GPRFC; it also seems intuitive that combining multiple source features means a created r.f is inherently less likely to be misleading, especially as the datasets are generated in an unsupervised manner. The DT results (Figure 5.6b) show no clear difference between the methods, though, as before, the DT method uses a higher proportion of the features in GPMVRFC than in GPRFC.



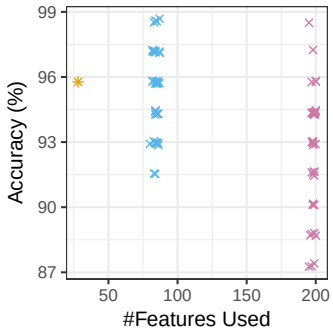
(a) Features ranked by IG, classified with KNN.



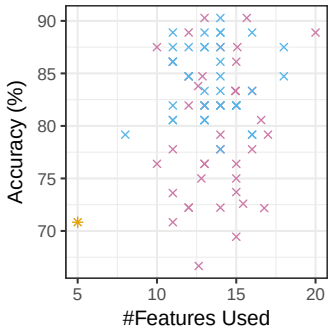
(b) Classified with DT.



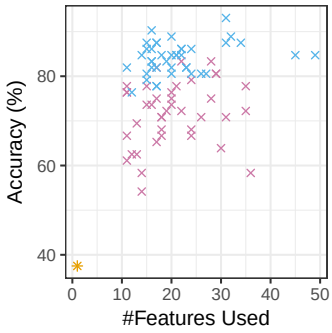
(c) FS with SFFS, classified with KNN.



(d) FS with SFBS, classified with KNN.

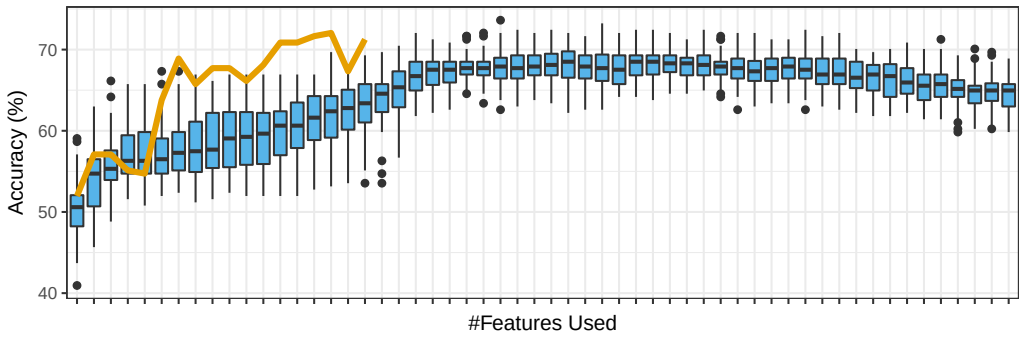


(e) FS with l1SVC, classified with KNN.

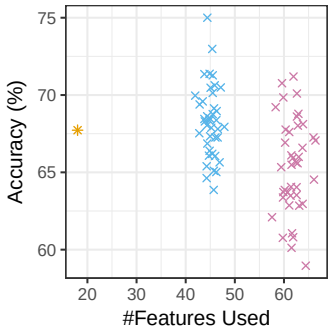


(f) FS with JMI, classified with KNN.

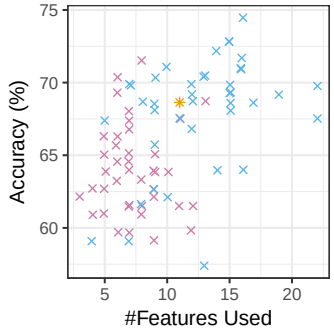
Figure 5.6: Dermatology Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is the existing GPRFC method.



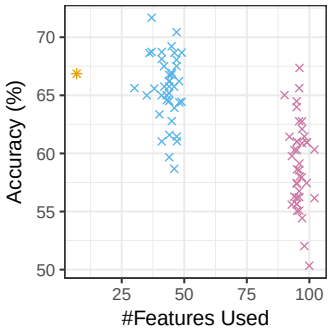
(a) Features ranked by IG, classified with KNN.



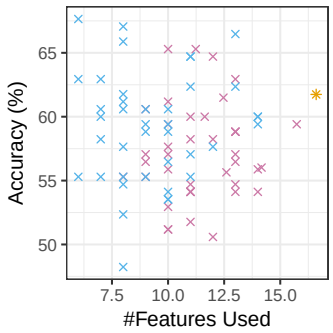
(b) Classified with DT.



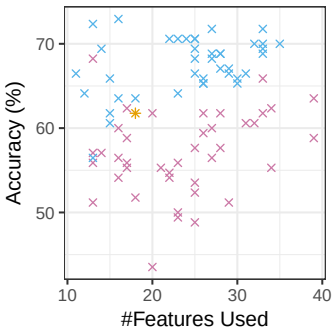
(c) FS with SFFS, classified with KNN.



(d) FS with SFBS, classified with KNN.



(e) FS with l1SVC, classified with KNN.



(f) FS with JMI, classified with KNN.

Figure 5.7: Vehicle Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is the existing GPRFC method.

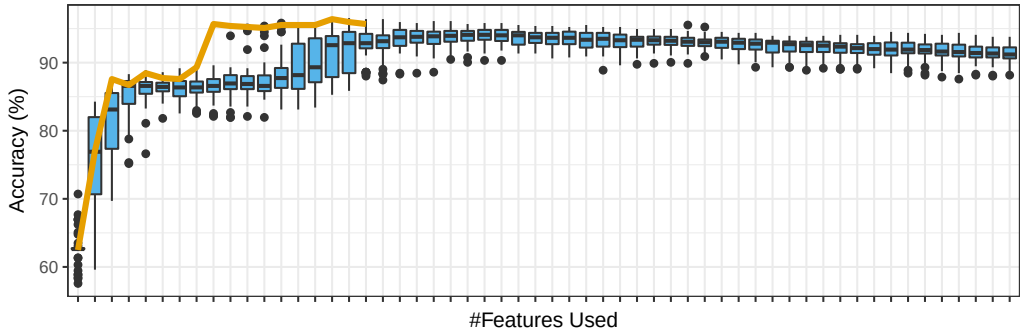
5.8.3 Vehicle

The Vehicle dataset results (Figure 5.7) continue to show a similar pattern. The primary exception is feature ranking (Figure 5.7a), where the GPMVRFC-augmented datasets clearly decrease the performance of

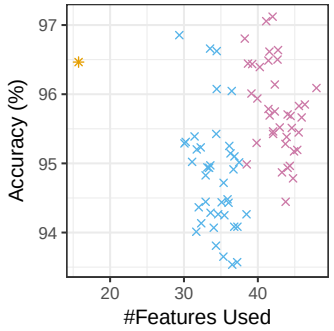
the classifier. As discussed before, this may be due to feature ranking being a univariate approach that cannot cope with multivariate feature interactions. Interestingly, the r.fs also cause the performance of the classifier to *decrease* when the last third of features are used, suggesting some features may be misleading to the KNN classifier. Each of the remaining results (Figures 5.7b to 5.7f) show that GPMVRFC can clearly create better features than those in the original dataset, causing the FS methods to often select additional, better features, except for l1SVC, which actually selects fewer features with better performance on occasion. The DT and SFBS show very consistent patterns in this case, perhaps as they both start by using the whole feature set, before removing unhelpful features.

5.8.4 Image Segmentation

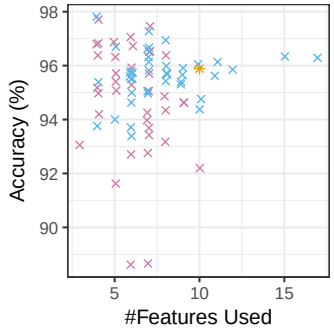
On the final dataset, the general pattern in the results (Figure 5.8) is that GPMVRFC produces r.fs that cause **lower** test accuracy than GPRFC, despite GPRFC producing twice as many r.fs. The feature ranking method clearly suffers in accuracy, including a slow drop off in accuracy even after only half the best-ranked features have been used (Figure 5.8a). The SFBS, l1SVC, and DT methods (Figures 5.8b, 5.8d and 5.8e) all select significantly more features compared to on the original datasets while getting much lower test accuracy, clearly indicating the added r.fs are more challenging than those created by GPRFC. JMI (Figure 5.8f) under-selects features compared to the original dataset, or selects extra, less useful features; SFFS (Figure 5.8c) is the exception to the pattern in that it does not have clearly worse accuracy due to GPMVRFC, but it also selects many fewer features than the other FS methods and so may be less prone to over-fitting. We plan to investigate why GPMVRFC gave such different results on the Image Segmentation dataset further in the future.



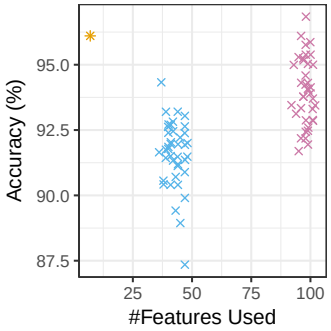
(a) Features ranked by IG, classified with KNN.



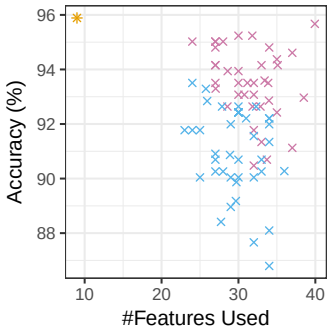
(b) Classified with DT.



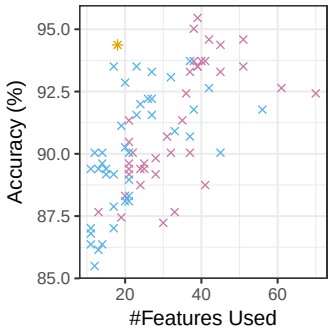
(c) FS with SFFS, classified with KNN.



(d) FS with SFBS, classified with KNN.



(e) FS with l1SVC, classified with KNN.



(f) FS with JMI, classified with KNN.

Figure 5.8: Image Segmentation Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is the existing GPRFC method.

5.9 Chapter Conclusions

This chapter introduced the first approaches to using GP for automatically generating difficult benchmark feature selection datasets containing complex redundant features.

The first approach, GPRFC, proposed a multi-tree representation and a novel mutual information-based fitness function. GPRFC was shown to generate high-quality and complex redundant features, which were suitable for augmenting existing datasets for use in testing feature selection algorithms. Testing of GPRFC showed the considerable potential of the use of GP for this task. We found that in order to improve GPRFC further, it would be necessary to develop an approach to creating more sophisticated and hard-to-detect multivariate redundant features. In addition, the fitness function showed limitations that could be improved.

These two factors were addressed in the second half of this chapter through the proposal of GPMVRFC. GPMVRFC used a refined approach that encodes multivariate features, and a significantly more appropriate and efficient fitness function utilising an elegant gradient-based technique. A large number of experiments were conducted to evaluate the proposed GPMVRFC approach, utilising a range of datasets and feature selection techniques. Several interesting patterns observed in the results were analysed in-depth, which showed that often GPMVRFC was able to produce challenging benchmark datasets that caused a variety of problems for different feature selection methods.

In some scenarios, it was found that GPMVRFC actually improved classification accuracy compared to when FS was performed on the original datasets. We believe this may be the result of the proposed multivariate approach inadvertently creating better features due to utilising multiple features from the original dataset. While this at first appears to be a negative result, it actually suggests something much more interesting: perhaps it is possible to perform unsupervised feature

construction with GP in a filter-based approach using a MI-based fitness function. This finding alone clearly warrants future investigation.

This chapter clearly showed the potential for GP-based FC to be applied to a new and relatively unstudied research direction. Manifold learning is an unsupervised learning research area that has significantly increased in popularity in recent years, but has never seen the use of GP-based FC. Many state-of-the-art manifold learning methods are very difficult to interpret, despite manifold learning being primarily used in data exploration tasks. The next chapter will investigate the potential for GP-based FC to improve the interpretability in manifold learning, providing further evidence of the potential for evolutionary feature manipulation to improve the outcomes of unsupervised learning tasks.

Chapter 6

Interpretable Manifold Learning using Genetic Programming

6.1 Introduction

Manifold learning (MaL) has surged in popularity in recent years due to new techniques such as t-Distributed Stochastic Neighbourhood Embedding (t-SNE) and autoencoders. However, despite one of the core aims of MaL being to reduce complexity and improve data understanding, state-of-the-art MaL techniques produce opaque results with no clear mapping from the original data.

GP is well-known for producing *functions*, which map inputs (the domain) to outputs (the codomain) using tree-based structures [135]. GP appears to have several promising characteristics for solving this problem:

- It is a global learner, and so should be less prone to producing partial-manifolds (i.e. local minima) unlike many existing methods that use gradient descent or other approaches;
- As it uses an evolutionary-based search method, it does not require a differentiable fitness function (unlike auto-encoders, t-SNE, etc.) and so could be used with a range of optimisation criteria; and

- It is intrinsically suited to producing interpretable mappings, as tree-based GP in particular can be understood by evaluating the tree from bottom to top. A wide range of tools are also available for producing interpretable GP models, including automatic program simplification and parsimony pressure.

Despite these traits, there is no work that uses GP to learn a manifold by mapping an input dataset to a set of lower-dimensional outputs.

6.1.1 Chapter Goals

This chapter aims to propose the first GP approach to performing interpretable manifold learning by automatically evolving functional mappings from a high-dimensional space to a low-dimensional manifold. Such an approach is expected to perform high-quality dimensionality reduction using understandable models that provide clear insight into manifold structure. This chapter will investigate:

1. whether a multi-tree GP representation with specially designed function and terminal sets can be used for performing manifold learning;
2. if an appropriate fitness function can be developed to evaluate how effectively a GP individual preserves the structure of the high-dimensional space;
3. whether such an approach can compete with existing manifold learning algorithms while also containing interpretable mappings;
4. if the above approach can be extended to produce interpretable visualisations (2D MaL) by finding a trade-off between quality and complexity of models using a multi-objective approach; and
5. whether such visualisations are competitive with existing manifold learning visualisation techniques, and if the interpretability of the GP models allows for greater understanding of the visualisations.

6.1.2 Chapter Organisation

Sections 6.2 to 6.5 address the first three goals of this chapter by introducing the first GP approach to directly perform manifold learning (GP-MaL). The success of this approach is evaluated by comparing its performance to a number of existing manifold learning methods across a range of manifold sizes on a variety of datasets. Sections 6.6 to 6.10 tackle the fourth, and fifth goals, by extending GP-MaL to focus on producing interpretable visualisations (i.e. 2D manifold learning), by using a multi-objective approach, which balances the two competing objectives of model size and visualisation quality. This method, called GP-tSNE, is compared extensively to the state-of-the-art visualisation method t-SNE, with a particular focus on analysing the trade-off between visualisation clarity and tree interpretability. A summary of the key findings of this chapter is provided in Section 6.11.

6.2 GP for Manifold Learning (GP-MaL)

The proposed method, GP-MaL, will be introduced in three stages. Firstly, the design of the terminal and function sets is discussed. Then, a fitness function appropriate for manifold learning is formulated and explained. Finally, a method to improve the computational efficiency of GP-MaL (while maintaining good performance) is developed.

6.2.1 GP Representation

GP-MaL utilises a multi-tree GP representation, where each tree represents a single dimension in the output (low-dimensional) space. While multi-tree GP is known to scale poorly as the number of trees (t) increases, manifold learning usually assumes a low output dimensionality (e.g. $t < 10$). The terminal set consists of the d scaled real-valued input features, as well as random constants drawn from $U[-1, +1]$ to allow for variable sub-tree weighting. The output of each tree is not scaled or normalised in any way as this may introduce bias to

Table 6.1: Summary of the function set used in GP-MaL.

Category	Arithmetic			Non-Linear		Conditional		
Function	$+$	$\times$	$5+$	Sigmoid	ReLU	Max	Min	If
No. of Inputs	2	2	5	1	1	2	2	3
No. of Outputs	1	1	1	1	1	1	1	1

the evolved trees or affect tree interpretability.

The function set (Table 6.1) chosen is inspired by existing feature construction and manifold learning literature. It includes the standard "$+$" and "$\times$" arithmetic operators to allow simple combinations of features/sub-trees, as well as a "$5+$" operator that sums over five inputs[1] to encourage the use of many input features on large datasets. The commonly used subtraction and division operators were not included as they are the complements of addition and multiplication and so are redundant in the "way" in which they combine sub-trees. To encourage the learning of non-linear manifolds, two common non-linear activation functions from auto-encoders were added: the sigmoid and rectified linear unit (ReLU) operators. The function set also includes two conditional (non-differentiable!) operators, "max" and "min", which may allow GP to produce more advanced functions. Finally, the "if" function is also included, which takes three inputs a, b, c and outputs b if $a > 0$ or c otherwise, to allow for more flexible conditions to be learnt.

Mutation is performed by selecting a random tree in a GP individual, and then selecting a random sub-tree to mutate within that tree, as standard. Crossover is performed in a similar way, by selecting a random tree from each candidate individual, and then performing standard crossover.

6.2.2 Fitness Function

A common optimisation strategy among manifold learning algorithms is to encourage preserving the high-dimensional neighbourhood around

[1]Five inputs were found to be a good balance between encouraging wider trees and minimising computing resources required.

each instance in the low-dimensional space. For example, Multidimensional Scaling (MDS) attempts to maintain distances between points, whereas t-SNE uses a probabilistic approach to model how related different points are, and attempts to produce an embedding with a similar joint probability distribution. We refrain from using a distance-based approach due to the associated issues with the curse of dimensionality [42], and instead try to preserve the *ordering* of neighbours from the high to low dimensions.

Consider an instance I, which has ordered neighbours $N = \{N_1, N_2, ..., N_{n-1}\}$ for n instances neighbours in the high-dimensional space, and neighbours N' in the low-dimensional space. If we were to perfectly retain all structure in the dataset, then the ordering of N' must be identical to that of N, i.e. $N = N'$. In other words, the quality of the low-dimensional space can be measured by how *similar* N' is to N. In GP-MaL, we propose measuring similarity by how far each instances' neighbours deviate in their ordering in the low-dimensional space compared to the high-dimensional space. For example, if $N = \{N_1, N_2, N_3\}$ and $N' = \{N_2, N_3, N_1\}$, the neighbours deviate by 2, 1, and 1 positions, respectively. Clearly, the larger the deviation, the more inaccurately the orderings have been retained. Let $Pos(a, X)$ give the index of a in the ordering of X. We propose the following similarity measure:

$$Similarity(N, N') = \sum_{a \in N} Agreement(|Pos(a, N) - Pos(a, N')|) \quad (6.1)$$

where $Agreement$ is a function that gives **higher** values for **smaller** deviations. GP-MaL uses an $Agreement$ function based on a Gaussian weighting to allow for small deviations without significant penalty, while still penalising large deviations harshly. In this work, a Gaussian function with a μ of 0 and $\theta = 20$ is used. θ controls how harshly deviations are punished – in preliminary testing we found a high θ gave best results as it created a smoother fitness landscape. The weighting for

a given deviation dev is $1 - prob(-dev, +dev)$, i.e. the area of the Gaussian not in this range. In this way, when there is no deviation, the weighting is 1 (perfect), whereas when it is maximally deviated the weighting tends to 0.

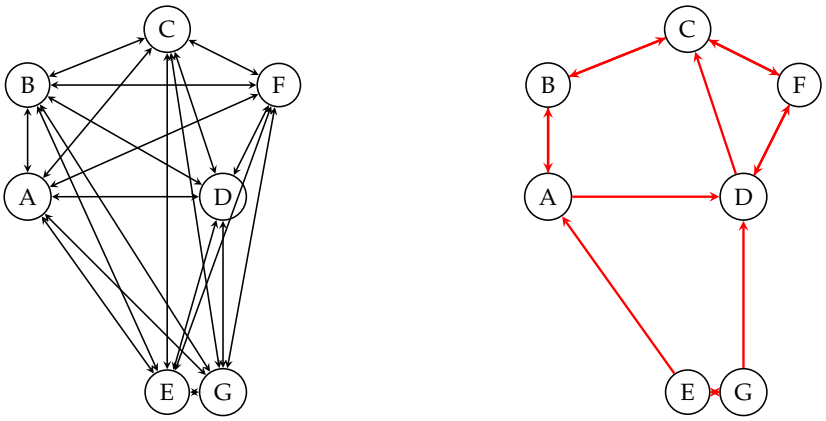
The complete fitness function is the normalised similarity across all instances in the dataset (X):

$$Fitness = \frac{1}{n^2} \sum_{I \in X} Similarity(N_I, N_I') \tag{6.2}$$

Fitness is in the range $[0, 1]$ and should be maximised.

6.2.3 Tackling the Computational Complexity

Computing the above fitness requires ordering every instance's neighbours by their distances in the low-dimensional space, at a cost of $O(n \log(n))$ using a comparison sort. This gives a net complexity of $O(n^2 \log(n))$ for *each* individual in the population. This scales poorly with the number of instances in the dataset. Consider a given neighbour N_b, which comes after N_a and before N_c for some instance. Even if we do not optimise the deviation of N_b, it seems likely that it will still be near N_a and N_c in the low-dimensional ordering, as it is likely to have similar feature values to N_a and N_c and hence will have a similar output from the evolved function. Based on this observation, we can omit some neighbours from our similarity function to reduce the computational complexity. Clearly, removing any neighbours will slightly reduce the accuracy of the fitness function, but this is made up by the significantly decreased computational cost (similar to surrogate model approaches [73]). An example of this can be seen in Figure 6.1, where the number of edges are decreased significantly by only considering the two nearest neighbours. Despite this, the global structure of the graph is still preserved well, with E and G being connected by only two edges to the rest of the (distant) nodes, and C, D, and F sharing many edges as they are in close proximity.



(a) Complete graph: 42 directed edges.

(b) Pruned graph: each node is connected to its two nearest neighbours. 14 edges.

Figure 6.1: Pruning of a graph to reduce computational complexity.

When considering which neighbours to omit, it is more important to consider the closer neighbours' deviations, in order to preserve local structure, as this is most likely to preserve useful information about relationships in the data. However, it is still important to consider more distant neighbours, so that the global structure is also preserved. Based on this, we propose choosing neighbours more infrequently the further down the nearest-neighbour list they are. One approach is to choose the first k neighbours, followed by k of the next $2k$ neighbours (evenly spaced), then k of the next $4k$, etc. This gives η neighbours according to the following equation:

$$\eta = k \log_2(\frac{n}{k} + 1) \tag{6.3}$$

thus η is proportional to $\log(n)$ ($k \ll n$). The complexity per GP individual is then $O(\eta \log(\eta)) = O(\log(n) \log(\log(n)))$, which gives a sublinear complexity. In preliminary testing, we found using $k = 10$ to give only minor differences in learning performance, which was significantly outweighed by the ability to train for many more generations in the same computational time. We use this approach in all GP-MaL experiments in this chapter. While k could perhaps be decreased further, it would not reduce computational time asymptotically, as tree

evaluation is now the main cost of the evolutionary process.

6.3 Experiment Design

To evaluate the quality of our proposed GP-MaL algorithm, we focus mainly on the attainable accuracy on classification datasets using the evolved low-dimensional datasets. High classification accuracy generally requires as much of the structure of the dataset to be retained as possible in order to find the best decision boundaries between classes, and so is a useful proxy for measuring the amount of retained structure. We refrain from using the fitness function (or similar optimisation criteria) to measure the manifold "quality" so as not to introduce bias towards any specific manifold learning method. The scikit-learn [129] implementation of the Random Forest (RF) classification algorithm (with 100 trees) is used as it is a widely used algorithm with high classification accuracy, is stable across a range of datasets, and has reasonably low computational cost [182]. While other algorithms could also be compared, we found the results to be generally consistent across algorithms, and so do not include these for brevity. 10-fold cross-validation is used to evaluate every generated low-dimensional dataset, and 40 evolved datasets (40 GP runs) are used for each tested dataset in order to account for evolutionary stochasticity.

The characteristics of the ten datasets we used for our experiments are shown in Table 6.2. A range of datasets from varying domains were chosen with different numbers of features, instances, and classes.

We compare the proposed GP-MaL method to a number of baseline manifold learning methods: PCA (as a linear baseline), MDS (which uses a similar optimisation criterion), LLE (a popular MaL method) and t-SNE (state-of-the-art for 2D/3D manifold learning). These methods are discussed further in Section 2.4 (page 43). Scikit-learn [129] was used for all the baseline methods (except for t-SNE) with default settings. For t-SNE, we used van der Maaten's more efficient Barnes-Hut

Table 6.2: Classification datasets used for experiments. Most datasets are sourced from the UCI repository [31].

Dataset	Instances	Features	Classes	Dataset	Instances	Features	Classes
Wine	178	13	3	COIL20	1440	1024	20
Move. Libras	360	90	15	Madelon	2600	500	10
Derm.	358	34	6	Yale	165	1024	15
Iono.	351	34	2	MFAT	2000	649	10
Image Seg.	2310	19	7	MNIST 2-class	2000	784	2

Table 6.3: GP Parameter Settings.

Parameter	Setting	Parameter	Setting
Generations	1000	Population Size	1024
Mutation	20%	Crossover	80%
Elitism	top 10	Selection Type	Tournament
Min. Tree Depth	2	Max. Tree Depth	8
Tournament Size	7	Pop. Initialisation	Half-and-half

implementation [170]. For each method and dataset, we produce transformed datasets for two, three, five, and the cube root ("cr") of the number of original features. Two/three features are useful for visualisation but are unlikely to be sufficient to preserve all structure, whereas the cube root approach was found in preliminary testing to be the point at which all tested methods could capture maximal structure from the datasets. Five features are used as a "middle-ground". As all of these implementations have stochastic components, we also ran each 40 times on each dataset.

We use standard GP parameter settings, as per Table 6.3. One notable setting is that we use 1000 generations; as we are interested primarily in exploring the *potential* of GP for this task, we are not particularly concerned with optimising the number of generations for best efficiency; this will be explored in future work.

6.4 Results and Analysis

The full set of results for each method and dataset are shown in Table 6.4. For each baseline method on each dataset, we label the result with a "$+$" if the baseline was significantly **better** than GP-MaL, or a "$-$" if it was significantly **worse**. If neither of these notations appear, there was no significant difference in the results. We used a two-tailed Mann-Whitney U test with a 95% confidence interval to compute significance. A summary of these results are provided in Table 6.5, by totalling the number of "wins" (significantly better), "losses" (significantly worse) and "draws" (no significant difference) the proposed GP-MaL method has compared with each baseline. We compare GP-MaL's performance to PCA and MDS, and LLE and t-SNE in the following subsections, as these pairs of methods exhibit similar patterns.

6.4.1 GP-MaL Compared to PCA & MDS

GP-MaL has a clear advantage over PCA when the most significant amount of feature reduction — to two or three features — is required. Given that PCA is a linear manifold learning method, it is not surprising that GP-MaL is able to preserve more structure in two or three dimensions by performing more complex, non-linear reductions. At higher dimensions, the gap narrows somewhat, as at five or cr features there are enough available output dimensions in order to make linear combinations able to model the underlying structure of the data more accurately. PCA weights every input feature in each component it creates, which means the way in which it models this structure is rather opaque when there are many input features. The MDS results have a similar pattern to the PCA ones, except that MDS and GP-MaL are quite even on three and five features. It is interesting to note that MDS uses a similar optimisation criterion to GP-MaL, but struggles more in creating a good two-dimensional manifold.

Table 6.4: Experiment Results. GPM refers to the proposed GP-MaL method. The number after each method specifies the dimensionality of the low-dimensional manifold; "cr" means the cube root approach determined the dimensionality.

Method	Wine	Move.	Derm.	Iono.	Image.	COIL20	Mad.	Yale	MFAT	MNIST
GPM2	0.955	0.485	0.914	0.826	0.797	0.628	0.605	0.382	0.639	0.909
PCA2	0.764−	0.405−	0.769−	0.776−	0.675−	0.647+	0.572−	0.244−	0.643	0.906−
MDS2	0.711−	0.476−	0.723−	0.837+	0.716−	0.732+	0.574−	0.339−	0.687+	0.909
LLE2	0.659−	0.499+	0.803−	0.833	0.809+	0.850+	0.601−	0.120−	0.843+	0.980+
tSNE2	0.718−	0.782+	0.852−	0.890+	0.921+	0.948+	0.712+	0.455+	0.935+	0.986+
GPM3	0.964	0.579	0.924	0.872	0.892	0.773	0.688	0.472	0.765	0.925
PCA3	0.793−	0.608+	0.780−	0.877	0.805−	0.823+	0.681−	0.374−	0.749−	0.932+
MDS3	0.726−	0.594+	0.774−	0.910+	0.883−	0.849+	0.677−	0.404−	0.830+	0.932+
LLE3	0.667−	0.513−	0.824−	0.847−	0.831−	0.923+	0.648−	0.297−	0.847+	0.984+
tSNE3	0.712−	0.768+	0.847−	0.756−	0.924+	0.952+	0.731+	0.394−	0.935+	0.987+
GPM5	0.960	0.673	0.951	0.915	0.958	0.847	0.864	0.553	0.888	0.940
PCA5	0.913−	0.705+	0.899−	0.923+	0.911−	0.887+	0.881+	0.531−	0.885	0.945+
MDS5	0.732−	0.719+	0.817−	0.928+	0.901−	0.886+	0.685−	0.564+	0.881−	0.948+
LLE5	0.683−	0.684+	0.825−	0.817−	0.837−	0.930+	0.665−	0.456−	0.870−	0.985+
tSNE5	0.718−	0.747+	0.835−	0.714−	0.930−	0.878+	0.763−	0.532−	0.939+	0.987+
GPMcr	0.962	0.681	0.941	0.899	0.891	0.913	0.863	0.661	0.935	0.952
PCAcr	0.789−	0.704+	0.852−	0.879−	0.804−	0.950+	0.857−	0.648−	0.939+	0.957+
MDScr	0.725−	0.722+	0.792−	0.920+	0.884−	0.911	0.670−	0.651	0.889−	0.957+
LLEcr	0.669−	0.685	0.814−	0.803−	0.828−	0.924+	0.679−	0.577−	0.912−	0.984+
tSNEcr	0.710−	0.759+	0.853−	0.713−	0.925+	0.730−	0.765−	0.650−	0.944+	0.987+

Table 6.5: Summary of Experiment Results. The number of "wins", "losses", and "draws" are shown for GP-MaL compared to each baseline.

Baseline	Wins	Losses	Draws	Baseline	Wins	Losses	Draws
PCA2	8	1	1	PCA3	6	3	1
MDS2	6	3	1	MDS3	5	5	0
LLE2	4	5	1	LLE3	7	3	0
tSNE2	2	8	0	tSNE3	4	6	0
PCA5	4	5	1	PCAc	6	4	0
MDS5	5	5	0	MDSc	5	3	2
LLE5	7	3	0	LLEc	7	2	1
tSNE5	6	4	0	tSNEc	6	4	0

6.4.2 GP-MaL Compared to LLE & t-SNE

Overall, GP-MaL is the most consistent of all the methods across the different numbers of features produced. LLE wins on one more dataset than GP-MaL for two features, but otherwise GP-MaL has a clear advantage with seven wins on three/five/cr features. The performance of LLE fluctuates quite widely across the datasets, and generally loses to PCA as the number of features is increased.

While GP-MaL is clearly worse than t-SNE on the 2 and 3D results, it outperforms t-SNE on five or cr features. On the Ionosphere and COIL20 datasets, t-SNE's performance actually decreases as the number of output features are increased, which means it is much more sensitive to the number of components that the user chooses than GP-MaL; GP-MaL almost strictly improves as more output features are produced, which is what we generally expect from dimensionality reduction techniques.

In a number of cases, t-SNE does actually outperform GP-MaL while using fewer features — however, consider that t-SNE (and LLE) are embedding methods, which do not have to manipulate the original feature space to produce the output feature space (i.e. are not a functional mappings). It is clearly more difficult to evolve such a mapping, but also has significant benefits in that GP-MaL's output dimensions can be interpreted in terms of how they combine the original features, which is often as important as visualisation alone in exploratory data analysis. This behaviour will be explored further in Section 6.5.

6.4.3 Summary

GP-MaL shows promising performance for an initial attempt at directly using GP for manifold learning, winning against all baselines on at least two of the four configurations tested. While GP-MaL faltered somewhat on some datasets such as MNIST, it achieved much better performance on other lower-dimensional datasets such as Wine and Dermatology. This suggests that with further improvements to its learning capacity, GP-MaL

may have the potential to outperform existing methods on these higher-dimensional datasets too.

Another important consideration is the interpretability of the models produced by each baseline. t-SNE, LLE, MDS, and PCA (to a lesser extent) are almost black-boxes as they give little information about how the manifolds in the data are represented in terms of the original features. Interpretability is an increasing concern in data mining, and feature reduction is often touted as a way to improve it; we will examine in the following section if GP-MaL can be interpreted any more easily than these existing methods.

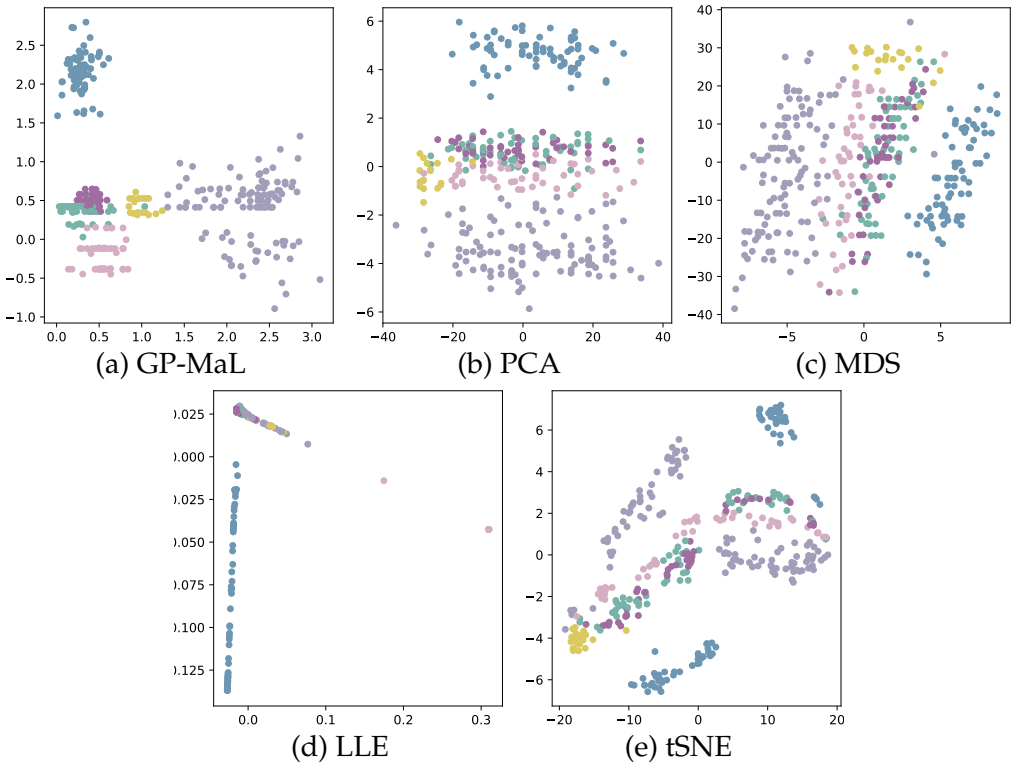
6.5 Further Analysis

6.5.1 GP-MaL for Data Visualisation

A common use of manifold learning techniques such as t-SNE and PCA is for visualisation of datasets in two- or three-dimensions. Figures 6.2 and 6.3 plot the two-dimensional outputs of each manifold learning method for the Dermatology and COIL20 datasets, which GP-MaL performed best and worst on, respectively. To show the potential of each method, we used the results that had the highest classification accuracy for plotting.

On the Dermatology dataset, GP-MaL clearly separates each class better than the baseline manifold methods. PCA, MDS, and t-SNE struggle to seperate the purple, green, and pink classes apart, whereas GP-MaL is able to separate them while keeping them reasonably close to signify their similarities. t-SNE splits both the purple and blue classes into two disjoint groups with other classes appearing in the middle of the split. LLE clearly struggles to give a good visualisation at all — it is only able to split the blue class from the others.

On the COIL20 dataset, LLE is able to separate the classes somewhat more effectively along one dimension, but still fails to produce a



(a) GP-MaL (b) PCA (c) MDS

(d) LLE (e) tSNE

Figure 6.2: The two created features on Dermatology, coloured by class label.

reasonable visualisation. t-SNE clearly does very well, but does continue to separate some classes into disjoint clusters (all of the green ones). It is not clear which of GP-MaL, PCA, and MDS produces the best result; MDS tends to incorrectly separate some classes like t-SNE, whereas GP-MaL and PCA have poorer separation of different classes overall. Unlike the other methods, the two dimensions produced by GP-MaL can be interpreted in terms of how they use the original features — this will be explored further in the next subsection.

6.5.2 Tree Interpretability

Part of an individual evolved on the MFAT dataset is shown in Figure 6.4. While the left tree is large (containing 73 features), the right tree is very simple: it adds two features together, and outputs the sum. This gives a

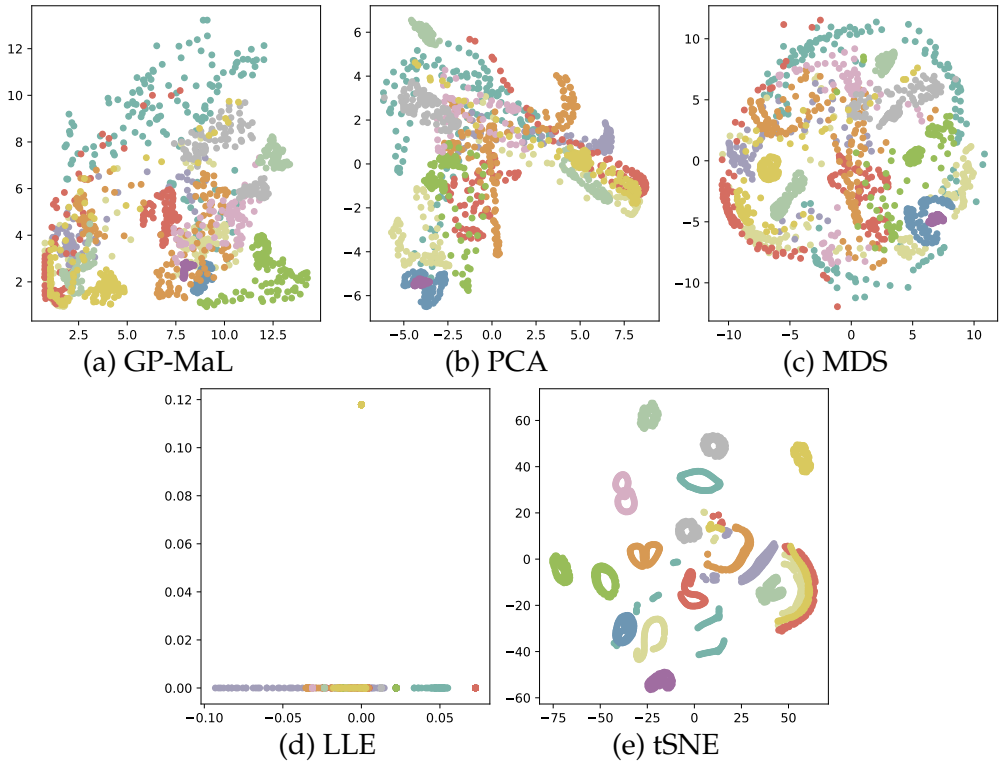


Figure 6.3: The two created features on COIL20, coloured by class label.

strong indication that these features are very important for modelling the underlying structure of the data. While the left tree is harder to analyse, it is useful to note that four of the children of the root node are again very simple: three features (two of which are transformed non-linearly) and a constant weighting. This again suggests that these features particularly model the instances in the MFAT dataset, with $X292$ appearing in both trees. $X292$ and $X294$ are the first and third Karhunen-Loève coefficients extracted from the original images; these coefficients are extracted in a similar way to PCA, so it makes sense that GP would recognise them as very useful features: being the first and third coefficients, they represent a significant amount of the variance present in this dataset.

Figure 6.5 shows an example GP individual evolved on the 500-dimensional Madelon dataset, with five trees used. Of the five trees, four are simple enough to be human-interpretable, with the fifth being

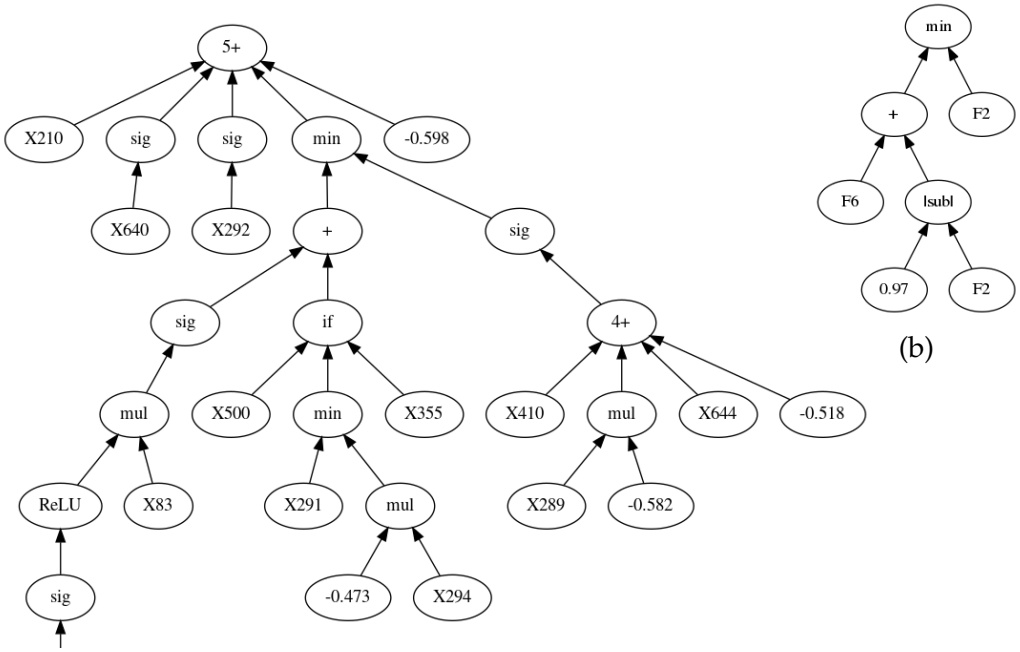


Figure 6.4: An example of two simplified trees (features) evolved on the MFAT dataset, giving 65% classification accuracy. Only the top of the left tree is shown.

larger, but still interpretable at the root. Trees 6.5b to 6.5e each combine between three and five features in an intuitive manner, but which would not be able to be represented by many existing manifold learning methods. For example, consider Tree 6.5c, which uses either the original value of $X475$, or a non-linear sigmoid transformation of $X475$ depending on the value of $X138$. This suggests that there is a particular feature interaction between $X138$ and $X475$ that may be important to the underlying structure of the dataset. In fact, $X475$ is the feature that has the second-highest information gain (IG) in this dataset. Tree 6.5e is just a single selected feature — $X455$ clearly is important in the manifold of this dataset.

Although Tree 6.5a is clearly more complex, the top of the tree still provides an interesting picture of the most important aspects of the dataset. For example, if $X48$ is a very low value, then this is simply the

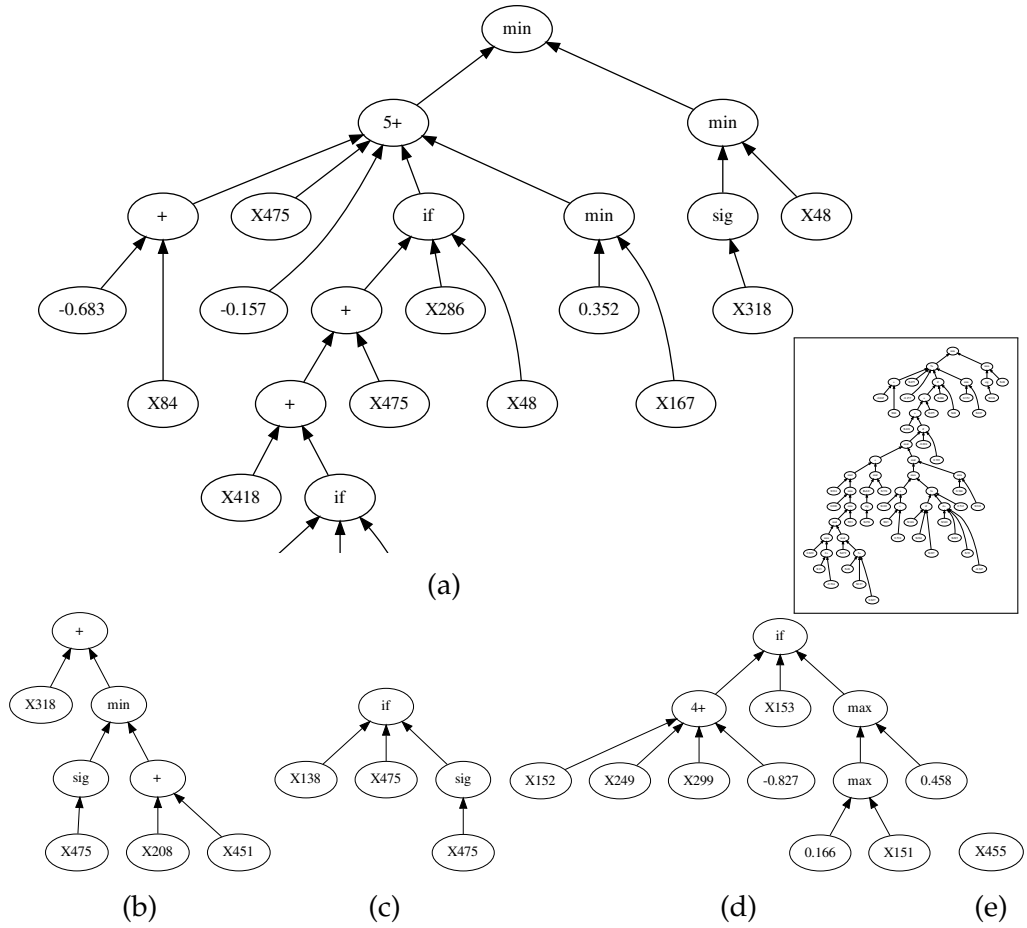


Figure 6.5: An example of five simplified trees (features) evolved on the Madelon dataset, giving 87.8% classification accuracy. Tree (a) was truncated to save space, but the full tree in the box to the right to show that is is comparatively small; the original dataset had 500 features.

output of the tree. Examining $X48$ more closely reveals that it is in the top 3% of features in terms of IG, and that at its smallest values it always predicts the positive class. Also of note is that $X475$ appears twice again in the top of this tree as well as in Tree 6.5c.

Summary:

As the focus of this work was to show the *potential* of GP for direct manifold learning, no parsimony pressure (or other such methods) were applied to encourage simple trees. Nevertheless, aspects of the evolved individuals can be analysed with ease and provide insight into the

structure of the datasets. This is a clear advantage over existing manifold learning techniques, which are black (or very grey) boxes, and bodes well for future work. The use of GP also has the benefit of allowing the evolved trees to be re-used on future examples without having to perform the whole manifold learning process again (as t-SNE or the other methods would require).

6.6 GP for Creating Interpretable Visualisations: GP-tSNE

The most successful application of manifold learning in recent years has been in visualisation tasks. State-of-the-art visualisation methods such as t-SNE use manifold learning to reduce dimensionality to two dimensions in order to preserve as much structure as possible, thereby making a highly accurate/representative visualisation. However, t-SNE, like most manifold learners, does not use a model-based approach to produce the two-dimensional representation, and so is very much a black box in terms of what the visualisation actually *means*. For example, if we find there are five "clusters" in a t-SNE visualisation, this is clearly useful information. But what we do not know is *why* the instances in these clusters are related — what characteristics of the original features makes them close neighbours? The remainder of this chapter extends GP-MaL to perform visualisation, focusing on making high-quality visualisations using GP models that are as simple and understandable as possible. Particular emphasis is put on the trade-off between visualisation accuracy and interpretability of models in terms of model complexity, through in-depth analysis that gives significant insight that is unattainable with existing black-box approaches.

6.7 Proposed Method: GP-tSNE

Generally when applying GP to a problem, there are two main decisions to be made: what terminal and functional nodes are suitable for the problem

(i.e. the architecture), and how to formulate a fitness function to solve the problem. In addition, there are often other improvements made to the standard GP evolutionary process to further tailor it to the problem. Each of these three considerations are discussed in the following subsections. While each of these components builds on established work, the use of them together to produce a range of interpretable models producing high-quality visualisations is a new and substantial advance in this field.

6.7.1 GP Representation

In order to project a high-dimensional space to a two-dimensional space for visualisation, it is important that a range of powerful and varied functions are available to the evolutionary process in order to produce compact but representative models. As exactly two dimensions are required for visualisation, we use a multi-tree approach where each individual contains two trees and each tree produces one dimension (i.e. the x- or y- axes) of the visualisation.

Table 6.6 lists the nine functions and four terminals used in the proposed GP-tSNE method. These are similar to those used by GP-MaL (Table 6.1), but with two key additions, which are highlighted below.

The four arithmetic functions are standard, with the exception of the $f+$ (flexible addition) function, which is the only function that can take the Zero node as input. This has the effect of allowing a variable number of inputs to this addition function, which allows the evolutionary process to easily perform mutation that effectively removes whole trees (hopefully *introns*[2]) in the pursuit of model simplification. The use of a Zero node is a key change compared to GP-MaL. Given that model simplicity is directly optimised in this work, it is important that GP individuals are easily able to add and remove sub-tree components; in GP-MaL introns were not particularly troublesome as the focus was on manifold retention. The $f+$ function replaces the $5+$ one in GP-MaL.

[2]Introns are "useless" sub-trees whose outputs do not affect the tree output.

Table 6.6: The function and terminal sets of GP-tSNE. $*$: Technically, the $f+$ function always has five inputs, but as the Zero node can be an input, semantically it can have fewer.

Function	No. of Inputs	Description
	Arithmetic Functions	
$+$	2	Std. Addition
$f+$	1–5*	Flexible Addition
$\times$	2	Std. Multiplication
$-$	2	Std. Subtraction
	Non-Linear Functions	
Sigmoid	1	$\frac{1}{1+e^{-x}}$
ReLU	1	$\max(0, x)$
	Conditional Functions	
Max	2	$\max(x, y)$
Min	2	$\min(x, y)$
If	3	if $(x < 0)$: y; else z
	Terminal Nodes	
F_i	0	i^{th} feature value.
$\mathrm{N}_j\mathrm{F}_i$	0	i^{th} feature value of neighbour j; $j \in [1, 3]$
Constant	0	From $U[-1, 1]$
Zero*	0	The number 0.

As in GP-MaL, the sigmoid and ReLU functions are included to allow easy transformation of (linear) inputs into a non-linear output space. Again, the three conditional operators provide different kinds of non-linear transformations, which are quite unique to GP due to their non-differentiability, and that are expected to allow a single tree to exhibit varied behaviour depending on its inputs.

In addition to the F_i terminal, GP-tSNE introduces the $\mathrm{N}_j\mathrm{F}_i$ terminal. This terminal gives the i^{th} feature value of the j^{th} neighbour of the instance being processed by the GP tree. This inclusion is based on the

observation that the feature values of a neighbouring instance in the high-dimensional space are likely to be useful for computing a point in the low-dimensional output space. The three nearest-neighbours for each point are pre-computed using Euclidean distance. While this terminal may also be useful in GP-MaL, it is particularly key in GP-tSNE as maintaining the local structure of a manifold is very important for visualising data accurately. As in GP-MaL, the constant terminal is included here with a range of $[-1, +1]$ to allow different parts of the tree to have different levels of impact on the final output. The Zero node is included solely for the $f+$ function, and is not used by any other functions[3] as it is either destructive or has no effect.

6.7.2 Multi-Objective Approach

There is an intrinsic trade-off in machine learning between the potential performance of a model and the minimum complexity required to attain that level of performance. For example, the simplest model to differentiate two classes would be a decision boundary that simply thresholds at a certain point in space, whereas for three linearly-separable classes, at least two thresholds would be required[4]. The same is true in visualisation: the more granular (specific) a visualisation is, the more complex the function used to produce that visualisation must be. To recreate the high-dimensional structure of a complex dataset in two dimensions would require two very big and complex GP trees. Each node removed from a tree decreases the accuracy with which the tree can reproduce the probability distribution from the high-dimensional space (in the case of t-SNE). As an analogy, consider evolving a very complex polynomial function with GP — the fewer components (nodes) in the evolved functions, the fewer inflection points available to approximate the function.

In this work, we employ a multi-objective approach to produce a set of

[3]The Zero node is *strongly typed* so it is only used by the $f+$ node.

[4]That is, for some real α, β: $Class_A < \alpha < Class_B < \beta < Class_C$.

solutions that allow for a trade-off between visualisation quality and model interpretability to be chosen. This is strictly a more difficult problem than optimising only visualisation quality (i.e. as in t-SNE) as a model must be found that maps the high-dimensional to the low-dimensional space. We choose to use NSGA-II [29] due to its wide acceptance and popularity on two-objective problems. The first objective uses t-SNE to measure the visualisation quality of an evolved GP tree, whereas the second objective uses tree size as a proxy measure for the interpretability of the evolved tree. Each of these will be discussed in turn.

6.7.3 Objective 1: Visualisation Quality

t-SNE uses conditional probabilities to represent the similarity between instances in a given dimensional space. Given two instances in the high-dimensional space, x_i and x_j, the conditional probability $p_{j|i}$ that x_j would be chosen as a neighbour of x_i is defined as [171]:

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq l} exp(-\|x_k - x_l\|^2/2\sigma_i^2)} \tag{6.4}$$

given a Gaussian with variance σ_i centred at x_i. t-SNE employs a symmetric approach where joint probability distributions are used; p_{ij} is computed as the symmetrised conditional probabilities, which for n instances is defined as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{6.5}$$

In order to remedy the crowding problem (see [171] for further details), t-SNE uses a slightly different approach in the low-dimensional space, which is based on a Student t-distribution. The joint probabilities of two instances, y_i and y_j, in the low-dimensional space, called q_{ij} is computed as:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \tag{6.6}$$

The cost function (C), which measures the extent to which the low-dimensional probability distribution does **not** match the high-dimensional probability distribution, is the difference between the two distributions as measured using the sum of the Kullback-Leibler (KL) divergences:

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{6.7}$$

We use this cost function as the first objective, which should be **minimized**. The parameter σ is computed based on a perplexity of 40 using the approach outlined in [171].

6.7.4 Objective 2: Model Complexity

A common issue encountered in GP is the production of bloated trees, where a GP tree is significantly bigger than is necessary to achieve a given level of fitness. Traditionally, there is no evolutionary pressure to encourage compact trees, and so trees may contain unnecessarily complex sub-trees, or in the worst case, introns. Bloated trees are computationally inefficient, but more importantly are also much harder for humans to interpret and understand.

Parsimony pressure, which treats minimisation of model size as a (minor) objective in the optimisation process, is the most frequently used method for controlling *bloat* in GP [107, 136]. Weighted sum approaches, where a small component of overall fitness is based on tree size, has been often used for attempting to control bloat [19], but choosing the right weighting is difficult and generally must be set empirically [155]. Lexicographic parsimony pressure, where the evolutionary selection process is modified to prefer smaller trees when fitnesses are nearly

equal [106], has also been frequently used to reduce the effect of bloat. More recently, multi-objective approaches have been proposed for addressing bloat, whereby tree size is used as a second objective to be minimised [176]. This approach allows a trade-off between fitness and model complexity to be found according to the approximated Pareto front produced by the GP process, while also producing a range of solutions of varying complexity in a single GP run, which can be compared by the user for better insight into the problem being tackled.

We use a simple formula for complexity in this work, which is based on the number of nodes in each of the two trees, T_a, T_b, in a GP individual I:

$$Complexity(I) = \sum_{T \in I} \sum_{N_i \in T} \begin{cases} 0, & \text{if } N_i = \text{ZeroNode} \\ 1, & \text{otherwise} \end{cases} \tag{6.8}$$

A given node N_i is counted towards the complexity unless it is a ZeroNode; these are not counted as they exist only to allow flexibility in the arity of the addition function, and can be removed from the tree structure when interpreting it as they have no semantic meaning.

6.7.5 Other Considerations

We use a slight variation to the standard NSGA-II, whereby we prevent the breeding process from producing two identical individuals in a single generation. Standard NSGA-II was found to often produce a new population containing a large number of the simplest individuals (i.e. tree size 1–2), which causes a collapse of the approximation front early on in the EC process. By enforcing this uniqueness constraint, a much more diverse and well-spread front is formed.

To further increase the visualisation quality without introducing additional model complexity, we use the covariance matrix adaptation evolution strategy (CMA-ES) [57] to fine-tune the numerical constants in each tree in the final front at the end of the evolutionary process. Standard GP has no ability to fine-tune its numerical coefficients

efficiently (as it effectively randomly searches the parameter space); by employing CMA-ES we can do so. CMA-ES is only used at the end of the evolutionary process both due to its computational cost and to prevent GP falling into local minima that would result from fine-tuning during evolution.

The evolutionary process was sped up through the use of multi-threaded evaluation, caching of quality values for previously-seen trees, and the use of linear algebra libraries such as Nd4j [34], which allows significantly faster matrix operations through native code libraries.

6.8 Experiment Setup: GP-tSNE

The proposed GP-tSNE method was applied to a range of representative datasets that contain well-formed classes that lend well to visualisation. The 10 datasets are summarised in Table 6.7, and have a range of numbers of instances, features, and classes. These datasets are from a number of different domains including general classification, biology, and image analysis. Most of these datasets were sourced from the UCI repository [31]. The standard t-SNE implementation [171] was chosen as a baseline method as it has the same optimisation measure as GP-tSNE, and is seen as the state-of-the-art in visualisation techniques. We used a standard perplexity value of 40 (the same as in GP-tSNE), and used the exact method of calculating nearest-neighbours. We also compare to GP-MaL as a "generic" GP-based MaL method.

We found that the use of a multi-objective approach necessitated a large number of generations (10,000) in the evolutionary process in order to allow the biggest trees (with the best performance) to be trained sufficiently. However, only a reasonably small population size of 256 individuals was needed to achieve reasonable coverage of the Pareto front, especially due to the restrictions used to prevent NSGA-II duplicating individuals. The remaining parameters, shown in Table 6.8 are standard settings; a maximum tree depth of 12 was found to be a

Table 6.7: Classification datasets used for experiments.

Dataset	Instances	Features	Classes
Iris	150	3	3
Wine	178	13	3
Dermatology	358	34	6
Breast Cancer Wisconsin	683	9	2
Vehicle	846	18	4
COIL20	1440	1024	20
Isolet	1560	617	26
MFAT	2000	649	10
MNIST 2-class	2000	784	2
Image Segmentation	2310	19	7

Table 6.8: GP Parameter Settings.

Parameter	Setting	Parameter	Setting
Generations	10,000	Population Size	256
Mutation	20%	Crossover	80%
Min. Tree Depth	2	Max. Tree Depth	12
No. Trees	2	Pop. Initialisation	Half-and-half

good trade-off between allowing complex and powerful trees while not increasing the search space and computational cost unreasonably.

All three methods were run 30 times on each dataset due to their stochastic nature. As our evaluation is primarily based on qualitative analysis (as is standard in visualisation [111,171]), we chose the median result out of the 30 runs according to the objective function used by the method: t-SNE's cost function for t-SNE and GP-tSNE, and GP-MaL's own fitness function for GP-MaL.

6.9 Results and Discussion: GP-tSNE

The results across each of the 10 datasets are shown in Figures 6.6 to 6.15. Each figure contains eight plots, which we refer to as (a)–(h) reading

left-to-right and top-to-bottom. Plot (a) shows the approximated Pareto front of the (median) GP result, with blue crosses showing each individual in the front, and the black line showing the shape of the front. The y and x-axes show each individual's value for Objectives 1 (visualisation quality) and 2 (model complexity) respectively. Plots (b)–(e) show representative visualisations — coloured by class label — produced by individuals at different levels of model complexity: (b) is the simplest possible model where each tree is a single feature (i.e. feature selection), (c) is the model at the lower-quartile of complexity, (d) is the median complexity model, and (e) is the most complex model, with the most accurate visualisation. The upper quartile is excluded as it is always very similar to the most complex model. Plot (f) shows the model that is closest to the origin (0,0) reference point on the approximated Pareto front, i.e. the model that could be considered to have the best trade-off between quality and model complexity based on scaled Euclidean distance to the origin. This model is also indicated on the approximated Pareto front in (a) by a red cross. Plots (g) and (h) are the t-SNE and GP-MaL baseline results with median performance, where the quality written above the plots is calculated using t-SNE's objective function for both, to allow for sensible comparisons. The tree size for GP-MaL is omitted as it was not optimised by GP-MaL and so is not indicative of the true complexity of models produced by GP-MaL.

6.9.1 Specific Analysis

On the simplest dataset, Iris (Figure 6.6), all three methods provide a clear 2D visualisation, although t-SNE and GP-tSNE more distantly separate the green class. The LQ result for GP-tSNE is quite similar to that produced by GP-MaL, but with less continuous spacing of points, likely due to the trees being too small to convert the low-granularity feature space to a more complex output space. The approximated Pareto front shows that diminishing returns are quickly found on this dataset: after ~25 tree size, quality only gradually improves as tree size quickly

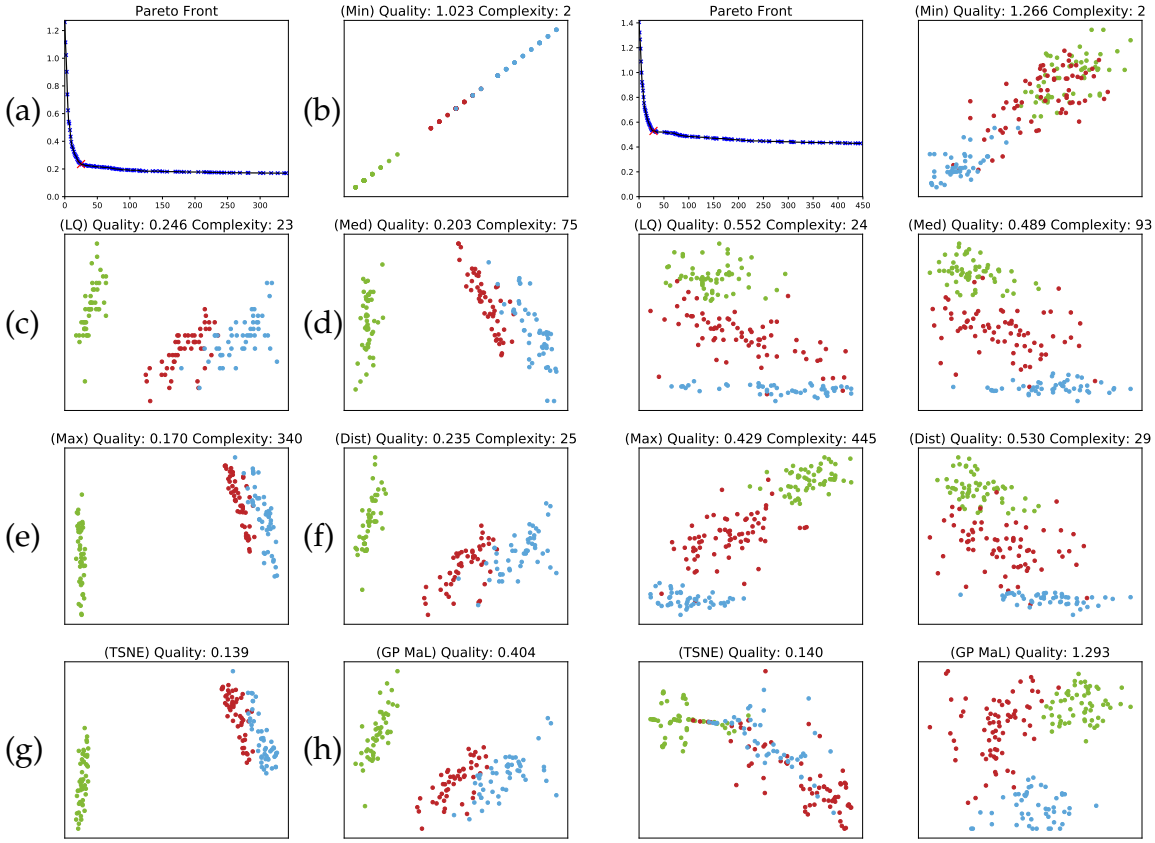


Figure 6.6: Iris.

Figure 6.7: Wine.

increases.

Despite achieving the best quality on the Wine dataset (Figure 6.7), the baseline t-SNE method seems to give a poorer visualisation compared to the two GP methods. t-SNE appears to put too much emphasis on maintaining the local structure of the data, such that the three classes begin to overlap significantly. Both GP methods showcase the three separate classes well, with the most fit GP-tSNE visualisation grouping each class compactly. Even at the LQ, GP-tSNE is able to start to separate the classes well compared to simple feature selection (two features).

Both t-SNE and GP-tSNE clearly separate the two classes on the Wisconsin Breast Cancer dataset (Figure 6.8), whereas GP-MaL does not give a clear separation boundary. A large model complexity is required

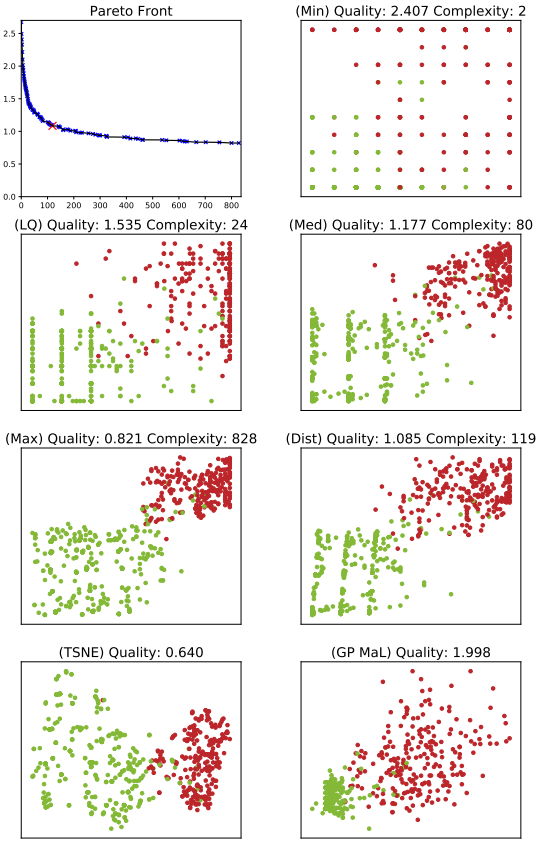


Figure 6.8: Breast Cancer Wisconsin.

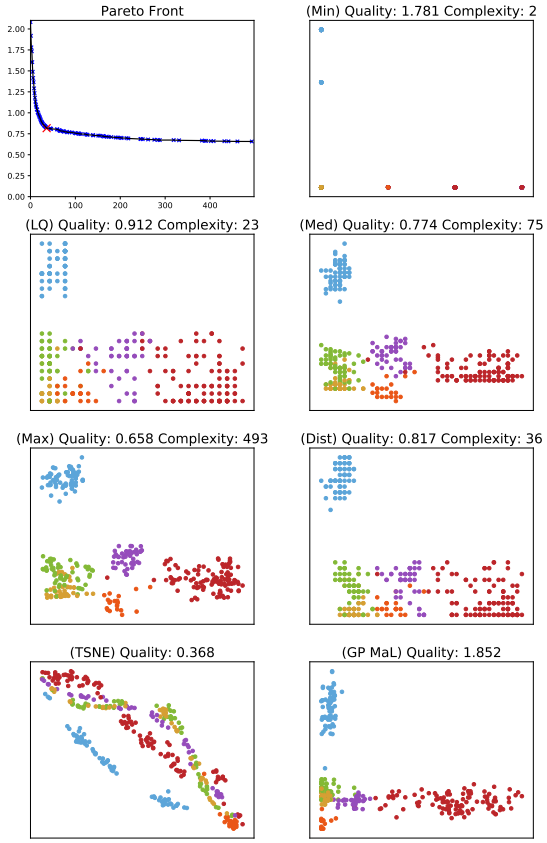


Figure 6.9: Dermatology.

for GP-MaL to remove artificial separations within the green class, though the global structure of the two classes starts to appear as early as a tree size of 80 (the median). Indeed, it appears that from the LQ through to the max tree size, the GP models retain the same general "approach", but become increasingly refined and accurate. In this way, it could be possible to use the simpler models to explore the more accurate patterns found by the bigger un-interpretable models by exploring how points move as complexity increases.

The Dermatology dataset (Figure 6.9) further shows this pattern: as GP-tSNE produces increasingly complex models, the visualisation becomes "higher-resolution" and produces points that appear to be in a continuous space, rather than a discrete space. GP-tSNE also clearly

produces the best visualisation on this dataset, despite t-SNE again having a better quality value. t-SNE artificially separates the blue class in two, while failing to separate the purple and red classes well at all. This suggests a possible unintended benefit of the proposed GP-tSNE approach: by forcing a functional mapping model to be used to produce the output, GP-tSNE is much less prone to artificially separating classes, as it cannot freely move points about the output space. GP-MaL is clearly worse than GP-tSNE, even when GP-tSNE has a tree size as low as 75.

Figure 6.10 shows the visualisations for the Vehicle dataset, one on which all three methods struggle to separate the four classes well at all. This suggests that the class labels may not provide a good representation of the underlying data. Some small patterns, such as the two groups of blue instances that are separated from the main set of instances, can be seen even at very low model complexity.

The COIL20 dataset (Figure 6.11) highlights t-SNE's ability to freely move points throughout the 2D space, as it clearly produces the best visualisation. It is interesting to note however that GP-tSNE is able to separate some classes well, even at low model complexity. For example, at the LQ, a number of classes are "clustered" quite well along the top of the visualisation. As the complexity increases, the lilac class becomes separable, and the curve topology of the blue/pink/red classes seen in the t-SNE visualisation starts to form for GP-tSNE too.

Figure 6.12 shows how the Isolet dataset has many overlapping classes, with only a few classes distinctly separated by t-SNE. The GP methods generally overlap the same classes as t-SNE, but do not manage to separate the groups as successfully. Isolet has the highest number of classes (26) of all the datasets, which makes it especially challenging for GP-based methods, since evolving two functions that can provide sufficient granularity to separate 26 classes is clearly very challenging and requires sufficiently complex trees. Despite this, it may be possible to gain some insight into why given classes overlap, as the general overlapping patterns start to be visible even at a low complexity of 65

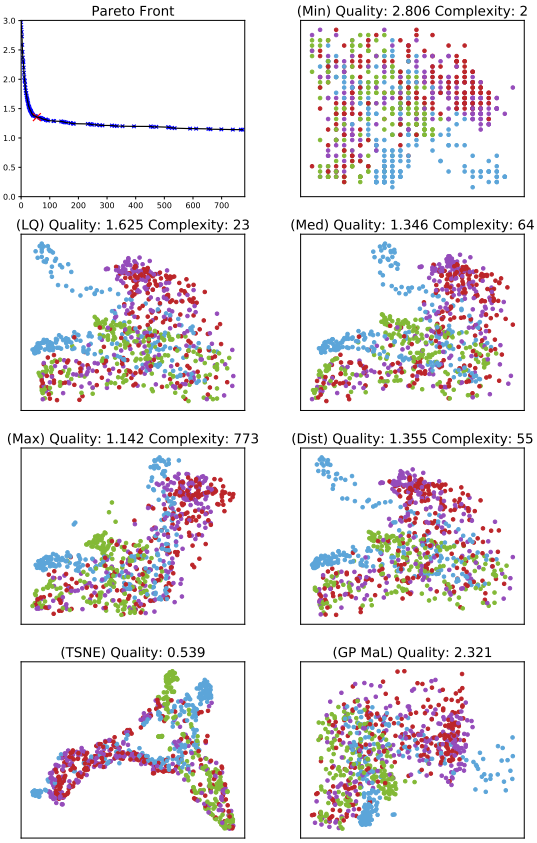


Figure 6.10: Vehicle.

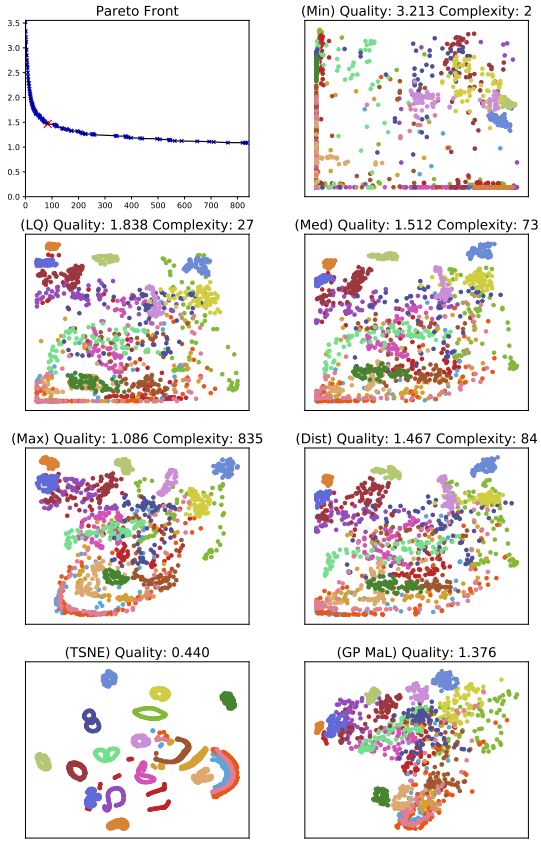


Figure 6.11: COIL20.

(the median model).

t-SNE performs quite well on the MFAT dataset (Figure 6.13), clearly separating six classes, but overlapping the other four significantly. The GP methods struggle somewhat, with GP-MaL not separating the classes well (especially the light green and purple). GP-tSNE manages to nearly separate the pink and yellow classes, and while it produces less class overlap than GP-MaL, the other classes do not have clear separation boundaries. The Pareto front is quite sparsely covered at the higher complexities, which suggests that the EC process did not focus enough on producing high-quality, complex models.

The visualisations in Figure 6.14 (MNIST 2-class) provide an example of a large dataset with a small number of classes. All three methods are able

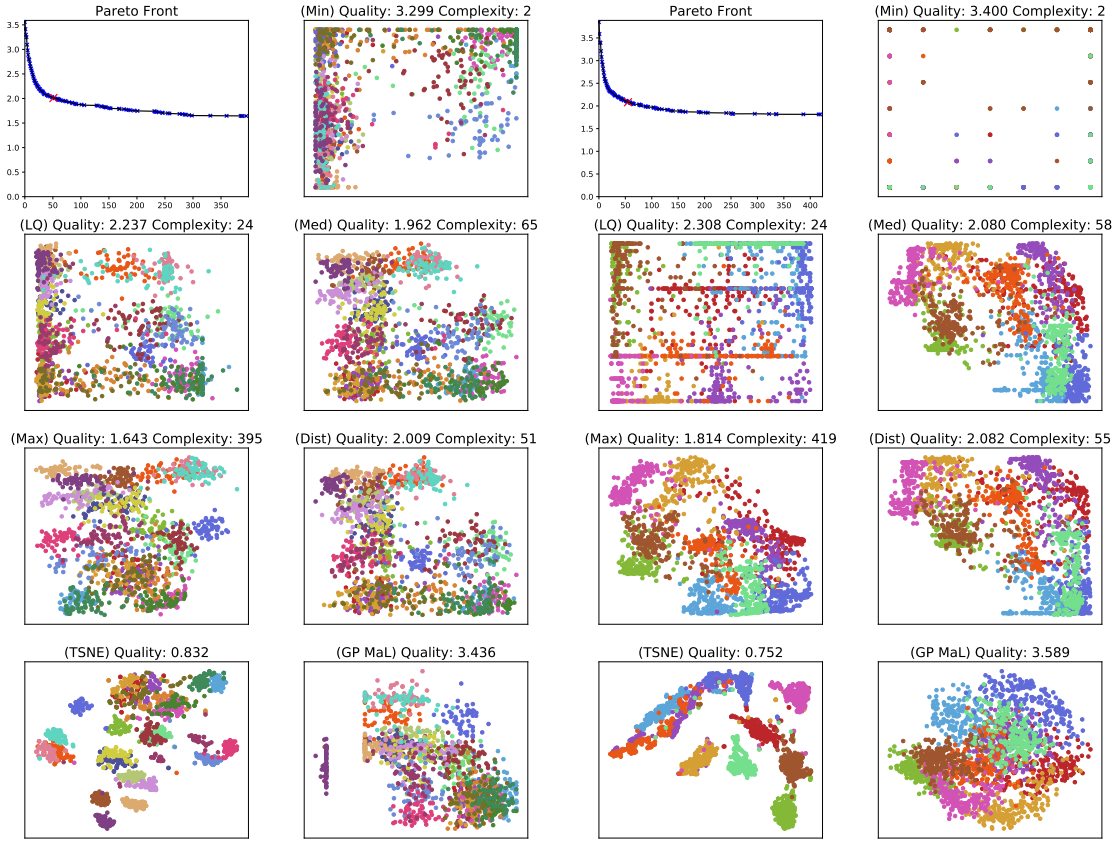


Figure 6.12: Isolet.

Figure 6.13: MFAT.

to separate the two classes reasonably well, with GP-tSNE and t-SNE producing quite similar results. This separation is evident even at low model complexity in GP-tSNE – at a complexity of 2 (feature selection), it is possible to draw a diagonal line through the middle of the visualisation that would separate the two classes with reasonable accuracy. GP-MaL produces a less clear separation and struggles to distribute the points in the visualisation space well.

On the final dataset, Image Segmentation (Figure 6.15), all three methods are able to separate the green and orange classes out, with t-SNE and GP-tSNE providing clearer separation margins. While t-SNE also manages to separate the red class, it does artificially separate parts of the blue, yellow, pink and purple classes. Conversely, GP-tSNE fails to

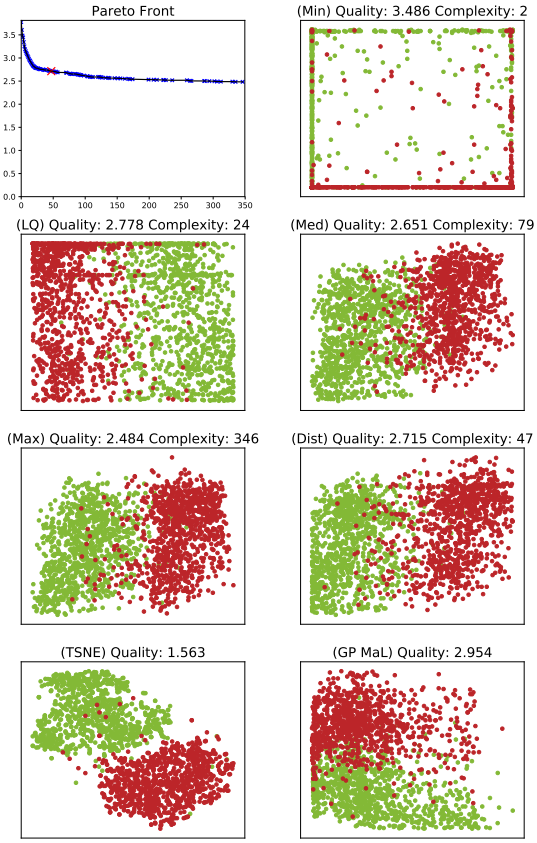


Figure 6.14: MNIST 2-class.

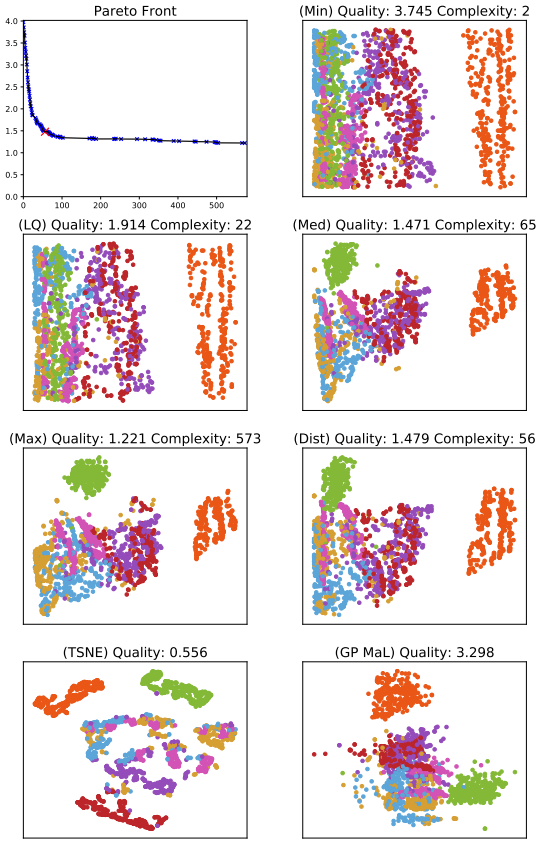


Figure 6.15: Image Segmentation.

separate the red class, but also keeps instances of the same class close to each other: the blue class appears in the same part of the large group rather than being split up as in t-SNE. Another useful observation is that the orange, and to a lesser extent the green classes, are separated at very low model complexity. The orange class is easily separated using only one feature, whereas the green class can be separated at a complexity of 65. This suggests that perhaps these are more natural/intrinsic classes in the dataset; perhaps they exhibit characteristics that make them particularly distinct from the other instances.

6.9.2 General Findings

The visualisations produced by GP-tSNE were consistently superior to those of our previous GP method, GP-MaL. GP-MaL was developed for more general manifold learning (i.e. not just reducing to two dimensions); by using a visualisation-specific quality measure, GP-tSNE is clearly better suited to visualisation. As the number of instances increased, t-SNE began to produce clearer visualisations than GP-tSNE, although the same general patterns were shown by both methods. Given GP-tSNE is in essence tackling a strictly more difficult task, of evolving a **functional model** to produce a visualisation, we see this as a success for the first such approach to this task.

One particularly interesting observation was how GP-tSNE produced similar overall visualisations at different model complexities, with the more complex models being more granular/refined than the simpler ones. This demonstrates the advantages of an evolutionary approach: good sub-trees of a complex tree can be used to improve simpler trees (and vice-versa) via crossover, allowing for more efficient search that produces models with different levels of trade-off. In the following section, we will explore this phenomenon in more depth to highlight the novel advantages of GP-tSNE.

6.10 Further Analysis: GP-tSNE

The results of GP-tSNE on the Dermatology dataset (Figure 6.9) were of particular interest as they clearly showed the potential of GP-tSNE; showed the same general visualisation, but at different qualities across tree sizes; and achieved good visualisations even at reasonably small model complexities. To further demonstrate the value of GP-tSNE, this section analyses a selection of increasingly complex trees produced by the median GP run on this dataset.

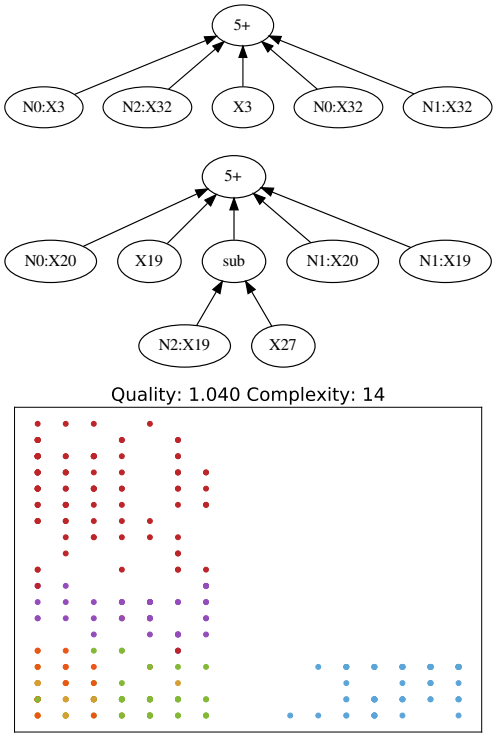


Figure 6.16: GP-tSNE at a complexity of 14 (Dermatology).

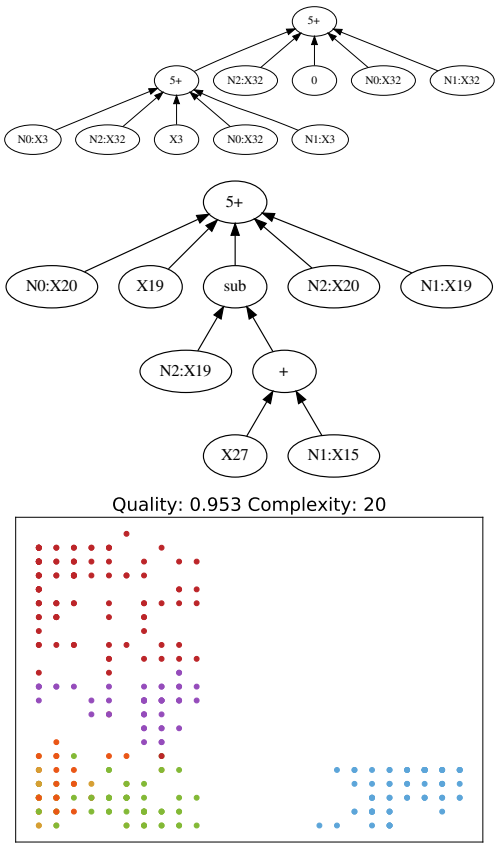


Figure 6.17: GP-tSNE at a complexity of 20 (Dermatology).

6.10.1 Simple Models

The simplest model that gives a reasonable visualisation is shown in Figure 6.16, which contains one tree with six nodes, and the other with eight, for a total complexity of 14. Even at such low complexity, four of the classes start to appear, with the blue class already showing a clear separation to the others. The first tree uses only two unique features: the instance's $X3$ value as well as its nearest-neighbour's $X3$ value, and the values of the three nearest-neighbour's $X32$ values. The second tree uses three unique features: the instance's $X27$ value; the instance's $X19$ and its two nearest-neighbours $X19$ values; and its two nearest-neighbours $X20$ values. This gives a total of five unique features of the 34 in the feature set. The blue class is separated from the other classes along the x-axis (the

first GP tree). The blue class is the third class in the Dermatology dataset, which is a type of skin rash called "lichen planus". The two features used by the first tree, $X3$ and $X32$, represent the amount of itchiness and the amount of band-like infiltrate affecting a patient's skin. A clinician can use this knowledge from the GP tree to realise that lichen planus can potentially be diagnosed based on these two symptoms alone.

When the model complexity is increased to 20 (Figure 6.17), the separation of points within classes starts to become better-defined. The first tree (x-axis) does not change significantly, except that the second nearest-neigbour's $X3$ value ($N1 : X3$) is added, and the $N0 : X32$ and $N2 : X32$ nodes are duplicated, to give a total summation of eight feature values, but still only two unique features. The duplication of these two nodes essentially doubles the effect of these features in the output of the tree. This suggests that these nodes are particularly important for improving the within-class separation along the x-axis. The second tree (y-axis) changes very little, with only the $N1 : X15$ node added, which causes a slight reshaping of the y-axis.

Figure 6.18 represents an increase of only one complexity to the previous model, but with a very different appearance. The second tree is actually identical to the first tree of the previous model but has swapped position, causing an inversion of the visualisations from here on in. The first tree is surprisingly different to the second tree of the previous model, as it adds features $X21$ and $X30$ and removes the use of features $X15$. This change is sufficient to give a clearer separation between the red and purple classes, and between the purple and green/orange classes, which suggests that features $X21$ and $X30$ may be particularly characteristic of the purple class.

Figure 6.19 further increases the amount of detail in the visualisation. The second model (y-axis) is still semantically unchanged from the first model of Figure 6.17. The first model (x-axis) however, now uses a total of seven unique features: $X6, 16, 19, 20, 21, 27, 29$. This allows it to spread the blue class out further, as well as adding clear distinction between the green and

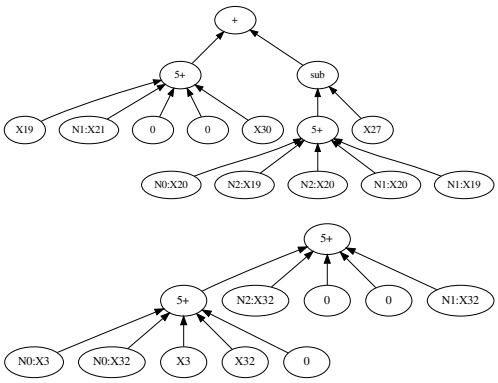


Figure 6.18: GP-tSNE at a complexity of 21 (Dermatology).

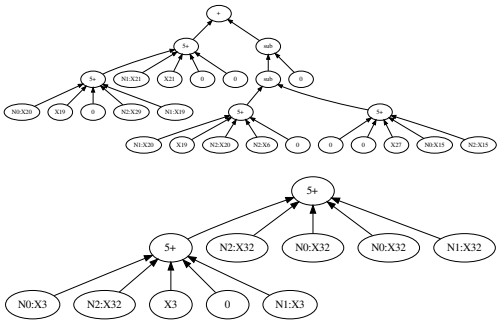


Figure 6.19: GP-tSNE at a complexity of 30 (Dermatology).

orange classes, which was not present before. The yellow class also begins to separate somewhat.

Interesting, all of Figures 6.16 to 6.19 use only linear functions (addition, subtraction), despite the wide range of functions available. A potential explanation is that at such low complexity, there is insufficient "room" in the tree to use non-linear functions in a sophisticated enough manner that would better separate the classes of the dataset. This is also quite beneficial in terms of these trees being easily interpretable, as linear functions are clearly simpler to understand than if conditional operators or sigmoid operators were used. Given that only linear functions are being utilised, another natural question is whether GP-tSNE is any better than PCA at this level of complexity — after all, PCA finds the best possible linear combination of features. However, PCA (in its primal form) has no ability to choose a small subset of features to combine,

whereas GP-tSNE uses only a few features, producing more concise and minimal models that are far simpler.

The sequential analysis of increasingly more complex models clearly provides additional insight that would be lacking in a non-multi-objective approach. This analysis technique is also an exclusive benefit of GP-MaL among other visualisation techniques, which are primarily black-boxes with one visualisation produced per run.

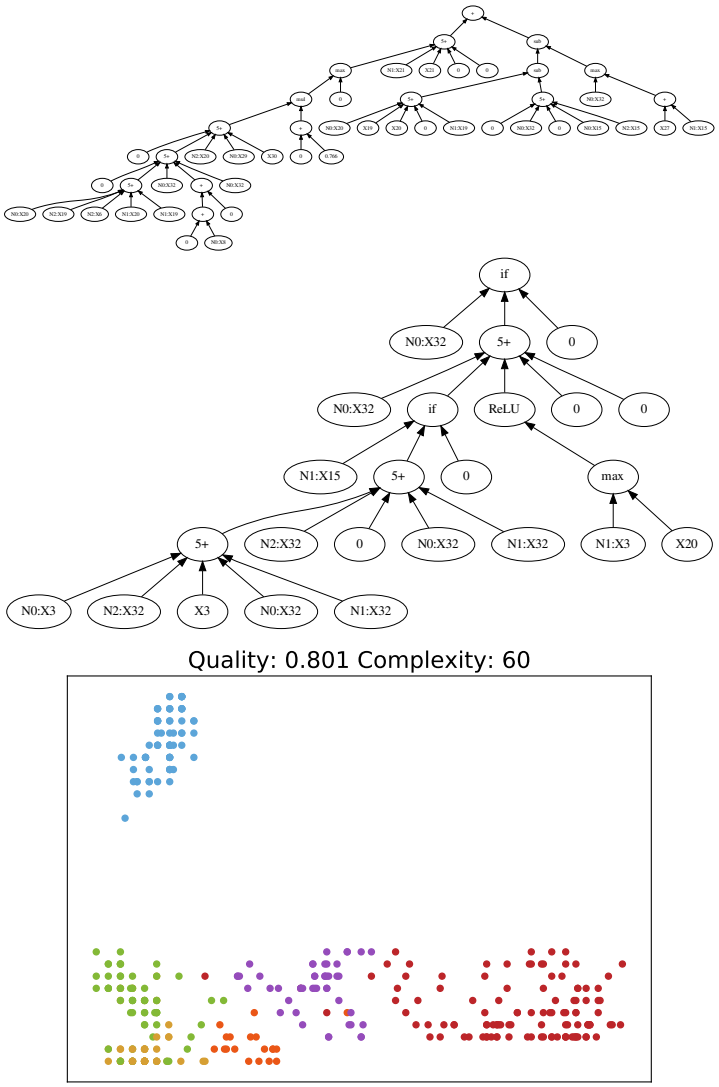


Figure 6.20: GP-tSNE at a complexity of 60 (Dermatology).

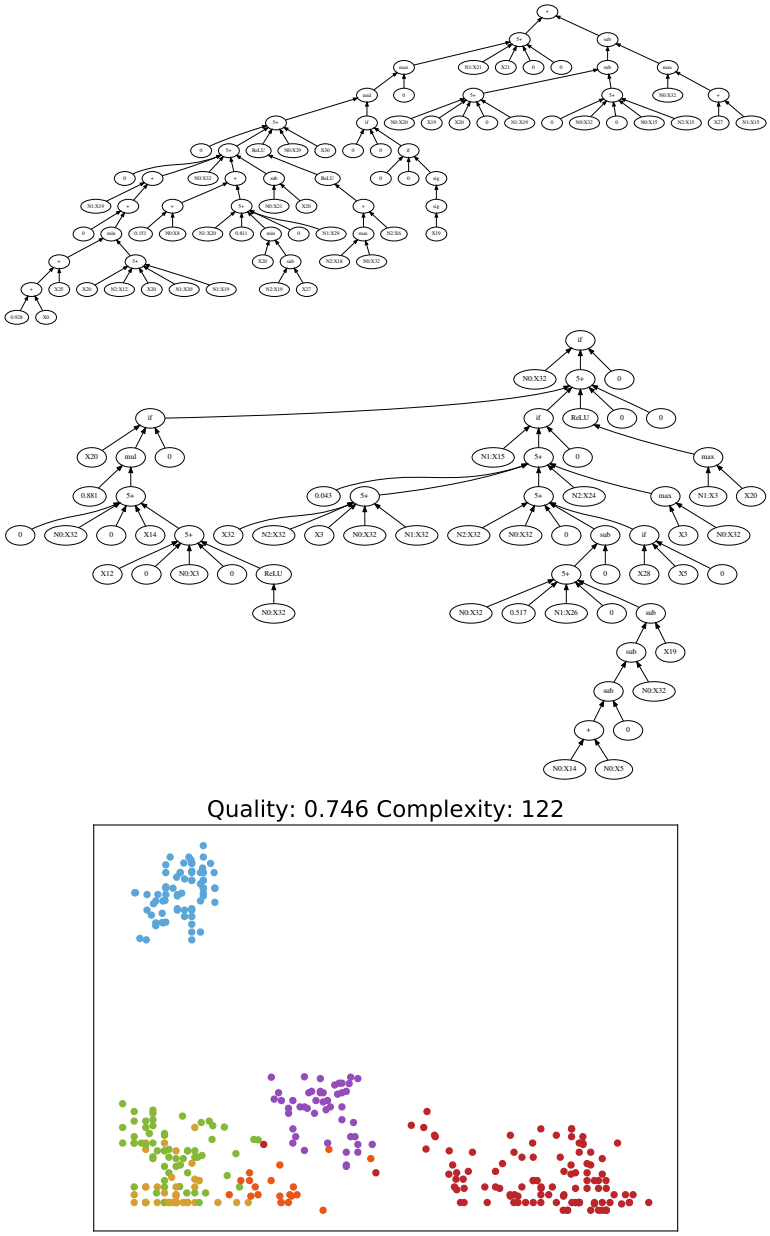


Figure 6.21: GP-tSNE at a complexity of 122 (Dermatology).

6.10.2 Complex Models

From here, the improvement in quality with the addition of more complexity begins to show clear diminishing returns: from a complexity of 14 to 30 gave an improvement in quality from 1.040 to 0.854, whereas

doubling the complexity again to 60 gives a further reduction only to 0.801, and to 122 a reduction to 0.746. These two models (Figures 6.20 and 6.21) do not fundamentally change the class separations present in Figure 6.19, but better separate within-class instances, as well as moving the purple class away from the other classes. Figure 6.21 also begins to separate the orange class. The GP trees in these models are also difficult to interpret — at a complexity of 60, the second tree is still quite simple, with only four unique features used ($X3, 15, 20, 32$), but the first tree uses nine unique features. These are also the first trees examined so far that use functions beyond arithmetic operators: earlier models achieved best results through simple addition and subtraction, whereas these trees use non-linear transformations (multiply, ReLU, sigmoid, max).

By the time that the maximal tree size (and best visualisation quality) is reached in Figures 6.22 and 6.23, the trees are nearly un-interpretable. The main change in the visualisation is the separation of the orange class to the bottom of the y-axis, as well as a more compact and separated purple class. This suggests that the "pityriasis rubra pilaris" skin disorder represented by the orange class is difficult to diagnose, and requires complex decision making based on many features to isolate clearly.

6.10.3 Summary

In this section, we showed how progressive examination of the approximated Pareto front from least to most complex models allowed insight into the structure of the data that is not easily achievable through traditional visualisation algorithms. Using the Dermatology dataset as a case study, we showed that even simple models could separate out some classes clearly, with the use of only two features. As we increased the complexity, we found that the granularity/local structure within classes improved, but the overall patterns changed only slowly. In this way, it is possible to use simple, understandable models to provide insight into how more complex models are able to produce high-quality visualisations.

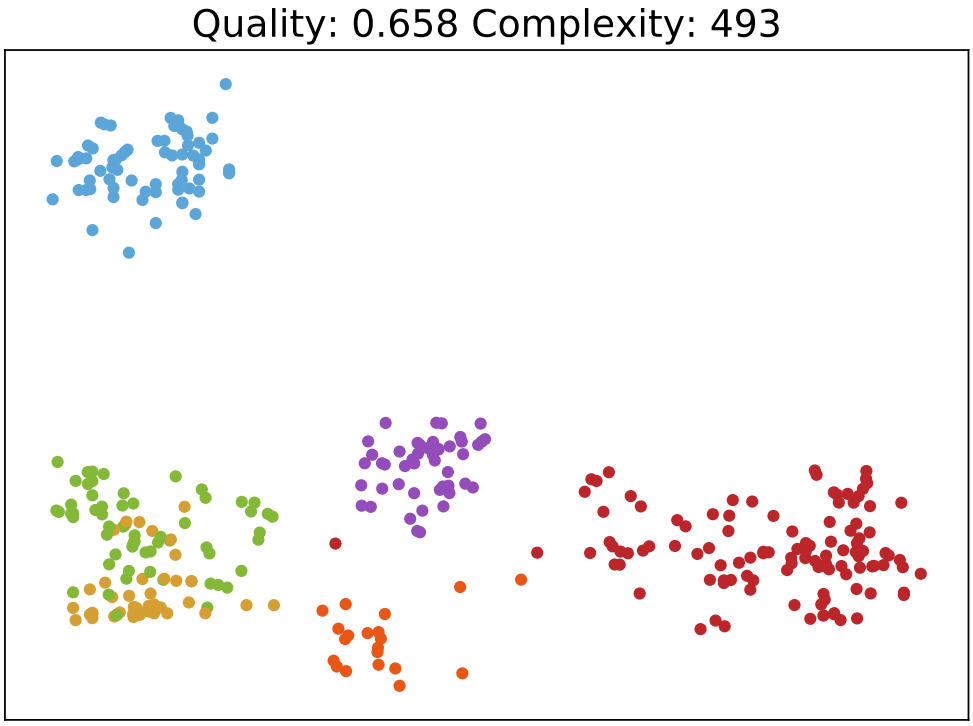


Figure 6.22: GP-tSNE at a complexity of 493 (Dermatology): Visualisation.

6.11 Chapter Conclusions

This chapter explored the potential for GP to be used for interpretable manifold learning through the development of two methods: GP-MaL and GP-tSNE.

GP-MaL was proposed as a generic manifold learning method, capable of doing dimensionality reduction to an arbitrary number of output dimensions. Appropriate terminal and function sets were presented, along with a fitness function tailored to the task, and further techniques for reducing computational complexity. GP-MaL was shown to be competitive, and in some cases clearly better than, existing manifold learning methods, and was generally more stable across the datasets tested. GP-MaL was also shown to produce interpretable models that help the user to gain concrete insight into their dataset, unlike many existing manifold learning methods. The potential of GP-MaL for

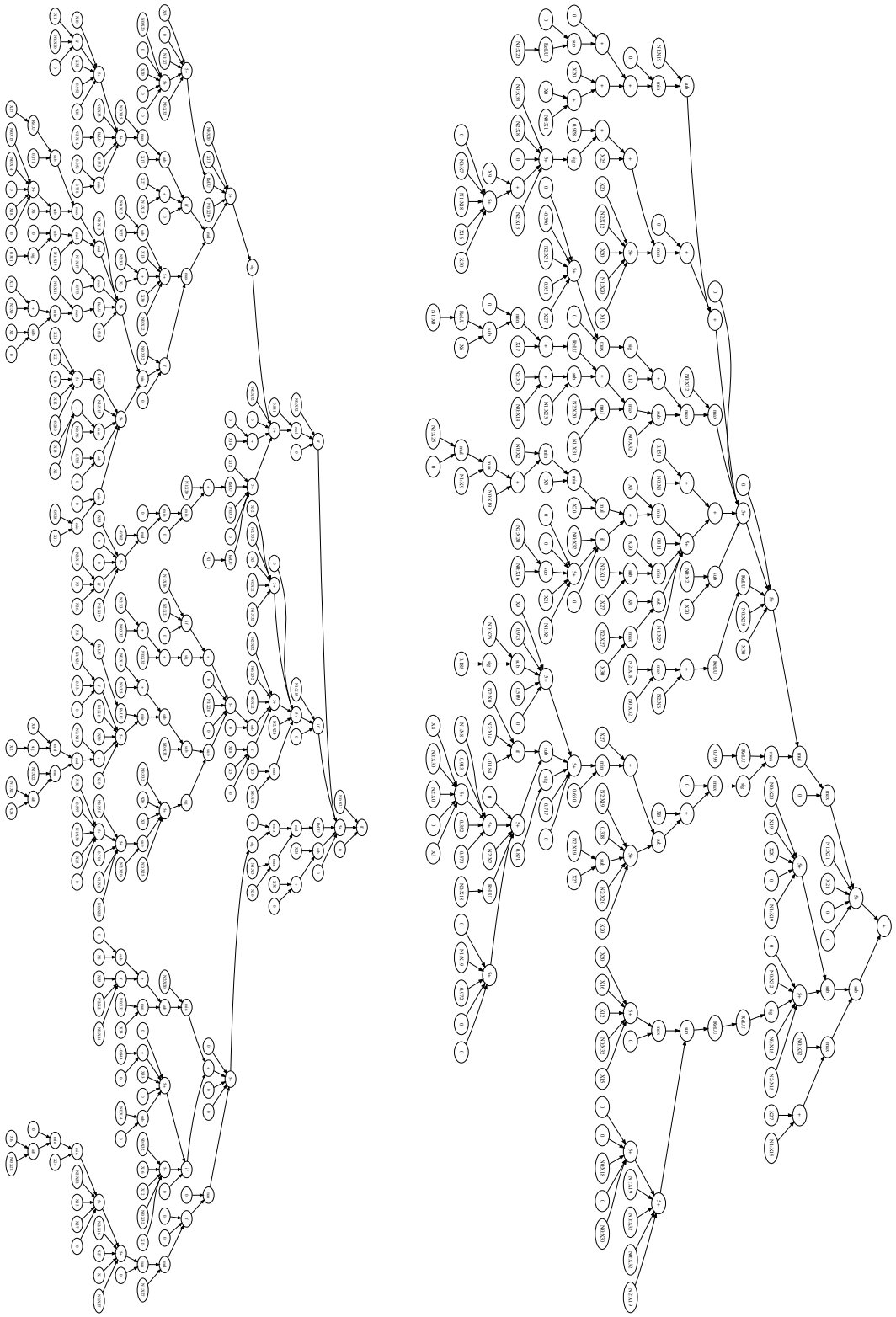


Figure 6.23: GP-tSNE at a complexity of 493 (Dermatology): Trees.

visualisation purposes was highlighted. Furthermore, the functional nature of the models produced by GP-MaL allows it to be applied to future data without re-training. As the first work using GP for directly performing manifold learning, these findings showcased the potential of GP to be applied further to this domain.

GP-tSNE extended GP-MaL significantly for specialised use as a visualisation algorithm. Visualisation is a fundamental task in data exploration that is used to provide insight into data that is too high-dimensional to be understood correctly, or that has not been collected for a particular purpose. GP-tSNE was motivated by the need for not only high-quality visualisations in data exploration, but for *interpretable* visualisations that can be understood in terms of the original features of the dataset. In addition to a refinement of the GP architecture and a visualisation-specific fitness function, GP-tSNE proposed a multi-objective approach to capture this critical trade-off between visualisation clarity and model complexity. Results on ten different datasets highlighted the promise in using a GP approach for this task, with visualisations produced that showed clear characteristics of datasets even at model complexities that could be low enough to understand. This was further showcased through an in-depth analysis of an evolved front on the Dermatology dataset, where concrete insight into how the dataset was structured was found by examining a range of models with different complexities.

Chapter 7

Conclusions

This thesis represents the first comprehensive investigation of applying EC techniques to perform feature manipulation (FM) in unsupervised learning. The overall goal was to improve algorithmic performance and model interpretability in unsupervised learning.

To achieve this goal, four main directions were investigated. First, EC-based FM was applied to the well-established and widespread unsupervised problem of clustering. A refined PSO-based approach for simultaneous clustering and FS was proposed that improved performance while selecting fewer features, even when the number of clusters was unknown. Secondly, two GP-based FC approaches were proposed that significantly improved clustering performance on a variety of datasets, while using only a small number of features in a way that could be understood by humans. These methods showcase the ability of EC-based FM approaches to improve results on common unsupervised learning tasks.

The third and fourth directions considers unsupervised learning problems where traditional algorithms have struggled, and that EC methods have not previously been applied to. By using novel EC-based FC approaches, this thesis proposes the first EC methods for automatically creating redundant features for benchmarking FS datasets. Finally, it also proposes the first GP-based methods for automatically

performing *interpretable* manifold learning. The results attained on these tasks highlight the potential for EC-based FM approaches to tackle difficult unexplored unsupervised problems in an interpretable manner.

This chapter will present the conclusions of each of the four research objectives, summarise the main findings of each contribution, and highlight promising future directions of research.

7.1 Achieved Objectives and Major Conclusions

7.1.1 PSO for Simultaneous Feature Selection and Clustering

The first objective of this thesis was to develop a new PSO-based algorithm to perform clustering and feature selection simultaneously in a single particle (Chapter 3). A medoid-based representation was proposed that allowed the number of clusters (K) to be automatically found within a fixed-length representation. A sophisticated three-stage extension was also proposed that improved the estimation of K; refined the fitness function further; and introduced a pseudo-local search to fine-tune clusters. Empirical evaluation showed clear benefits for the proposed approach, selecting fewer features than existing approaches while increasing clustering performance. This is the first PSO-based approach using a medoid representation for simultaneous feature selection and automatic clustering.

The Benefits of a Medoid Representation

The medoid representation — where each instance in a dataset can be selected as a cluster centre — had seen little use in the literature compared to the classic centroid representation. This thesis demonstrated the clear benefits to a medoid approach: cluster partitions produced by a medoid PSO method were better-structured while using fewer features than centroid PSO methods. Using a medoid approach constrains the

search space to only "sensible" solutions by forcing instances to be cluster centres. This improves the learning efficiency of PSO, and may allow the learning process to better focus on removing less useful features.

The medoid representation was found to have its own share of limitations. It has an intrinsic inability to fine-tune solutions due to the constraint of cluster centres needing to be instances. While this is useful during the *exploration* phase of PSO, it affects the *exploitation* that can be achieved. A psuedo-local search was proposed that converts the best particle of the run into a centroid representation, and then performs a small PSO run to fine-tune it. In this way, the benefits of both the medoid and centroid representations can be combined, while addressing the limitations of each.

Finding K

One key advantage of the medoid representation is its ability to produce a dynamic number of clusters with a fixed-length representation. However, using such a dynamic medoid approach introduces a new difficulty: the much larger search space introduced by allowing a variable number of clusters, as discussed in Section 1.2.1. Indeed, it was found that the dynamic medoid approach had significantly more trouble producing good solutions than the fixed medoid approach,which necessitated the introduction of a heuristic that produces an estimate of K, K_{est}. This heuristic was used to guide the PSO search, but still allowed flexibility in the chosen K, given the heuristic itself is imperfect. This significantly improved the accuracy of this method, allowing for a more useful algorithm that can automatically find the number of clusters, select features, and perform clustering.

Interaction between Feature Selection and Clustering

Performing clustering and feature selection *simultaneously* has the potential to produce a more cohesive set of features and clusters than if the two tasks are done *sequentially*. However, as discussed in

Section 1.2.2, there is a clear interaction between finding a small feature set and finding the most appropriate K. Chapter 3 clearly demonstrated this issue, and also showed that fewer features tended to be selected, at the expense of clustering performance as the feature selection task gave a much smoother/easier fitness space to optimise. The three-stage dynamic medoid approach addressed this limitation by exploring variations to the fitness function. Instead of linearly penalising fitness as the number of features selected increased, an elliptical weighting was proposed that punished a solution harshly only when the number of selected features was particularly large. Investigating an alternative multi-objective approach to balance these tasks may prove an interesting future direction of research.

7.1.2 GP for Feature Construction in Clustering

Chapter 4 introduced two new methods for using GP to perform FC to improve clustering performance, achieving the second objective of this work. Two wrapper-based FC approaches for clustering were proposed, improving clustering performance and interpretability by evolving a small set of high-level constructed features. The first method used a relatively direct approach, where constructed features (from GP trees) were fed into k-means clustering and the performance of the resulting clusters were used as the fitness of the constructed features. The second method (GPGC) utilised the functional nature of GP to evolve tailored similarity functions specific to the dataset (and algorithm) forming a clustering problem. Existing similarity functions, such as Euclidean distance, are designed with generality in mind: they are limited in the performance they can achieve and their behaviour on a specific dataset is very difficult to understand. The GPGC approaches (proposed in Section 4.6) directly addressed these limitations.

Benefits of Feature Construction in Clustering

While the benefits of using FC in supervised learning tasks are well known, this thesis provided clear evidence of the benefits in clustering as well. Many common clustering algorithms utilise rigid algorithmic approaches that mean they cannot automatically adapt to different domains or situations. By using a wrapper-based FC approach, the methods in this chapter are able to automatically tailor the search space as needed. By using operations such as *max*, *min*, and *if*, the constructed features can combine information across different features in a way that most clustering algorithms cannot. This allows for a much more flexible solution space, which allows the wrapped clustering algorithm to achieve higher performance than it would using only the original feature set.

Another key benefit to FC in clustering is the ability to construct multiple features that have heterogeneous behaviour across different clusters/parts of the feature space. Both the approaches in this chapter employed multi-tree approaches to produce multiple distinct constructed features using different sets of original features, which represent different characteristics of the dataset.

Interpretability of Cluster Partitions

Clustering, as a data exploration task, is generally used to give insight into datasets for which little information is known. For example, using clustering to find groups of related people may provide useful information about the customer base of a business. However, when data dimensionality is high, clustering becomes less meaningful even if high-quality clusters can be found. Using the same example, we would like to know not only *that* a group of people are related, but also *how* they are related. The FS approaches in Chapter 3 begun to show this, but the FC approaches in Chapter 4 highlighted this even more clearly. In Section 4.5, we explored how performance could be increased using 11 constructed features, which only used 12 unique original features in total

compared with the original 100-dimensional dataset. These constructed features were also very simple and easy to understand by examining the evolved GP trees.

The interpretability of the evolved tailored similarity functions were also highlighted in Section 4.10.1. An analysis of a multi-tree similarity function provided deep insight into how the clustering algorithm had decided to allocate different instances into different clusters. Only 15 features of the original 1000 were used, and performance was significantly better than when clustering methods used the standard Euclidean distance measure; this niching-style approach was able to give interpretable results while even improving cluster quality.

Chapter 4 clearly demonstrated the potential of GP for FC in clustering to not only improve performance, but also to improve the outcomes of the data exploration process by producing solutions that give clearer insight into the data.

7.1.3 Generating Benchmark Feature Selection Datasets with GP

The GPRFC and GPMVRFC methods proposed in Chapter 5 addressed the third objective by providing ways of automatically generating difficult uni or multivariate redundant features for creating benchmark FS datasets. GPRFC introduced a novel mutual information-based fitness function in conjunction with a multi-tree approach to create a set of (distinct) redundant features given a source feature. Experimental testing of GPRFC demonstrated the promise of GP for this task, but showed a clear need for a better fitness function and a way to create more complex multivariate redundant features. GPMVRFC is an extension of GPRFC designed to address both of these limitations, using a multivariate representation with a gradient-based fitness function to create more sophisticated redundant features. While GPMVRFC produced better results than GPRFC in some regards, it uncovered a particularly

interesting phenomenon: classification performance would actually often improve when the multivariate redundant features were added, despite them being created in an unsupervised manner.

Performing Unsupervised Filter-based Feature Construction using MI

In hindsight it seems somewhat obvious that creating multivariate redundant features would create high-quality features that could be directly useful for classification tasks. In order for a created feature to have a high MI with a set of source features, it must combine as much information from these features together as possible. If we assume that the classes in a dataset are reasonably well-structured, then it follows that such combined information is likely to be useful for predicting the class label. Furthermore, multiple distinct constructed features are created in tandem, further increasing the chance that at least one of these will contain useful information for a classification algorithm. This phenomenon is clearly a significant issue for the task of redundant feature creation; more importantly it offers an exciting future direction of research for creating powerful constructed features as an unsupervised pre-processing step that has the potential to be used to improve the performance on a diverse set of data mining tasks.

Ability of GP to Create Trees with Distinct Behaviour

One of the main challenges in developing these algorithms (in Chapter 5) was finding appropriate ways to encourage diversity across the created redundant features. GPRFC used a MI-based approach to reduce the redundancy between created features, with good success as seen in the created visualisations (shown in Chapter 5). These plots show that GP has clear potential to model a variety of complex functions by using a multi-tree representation with diversity pressure in the fitness function. This reinforces the findings discussed earlier in this chapter: heterogeneous behaviour across trees is a particularly distinctive and promising property of multi-tree GP, which could be exploited in a range

of data mining tasks. Minimising MI across GP trees in order to encourage this heterogeneity/niching is likely to be even more effective in other tasks, which do not otherwise employ MI in the fitness function.

7.1.4 Interpretable Manifold Learning using GP

The final objective of this thesis was to develop a GP-based approach to perform *interpretable* manifold learning. As previously discussed in Section 1.2.5 and Chapter 6, state-of-the-art manifold learning techniques achieve high performance at the expense of poor interpretability of the discovered low-dimensional manifold. The two methods proposed in Chapter 6 addressed this objective by introducing the first GP-based approaches to manifold learning, with a particular emphasis on the interpretability of the trees used to represent the discovered manifold.

The first method, GP-MaL, showed the potential of GP for performing general MaL (i.e. to any sized low-dimensional space). Despite being restricted by using a model-based approach, GP-MaL was often competitive with (or better than) existing MaL methods in the conducted experiments. More importantly, the ability of GP-MaL to produce interpretable manifold structures was demonstrated.

To further highlight the potential and practicality of using GP for MaL, an extension to GP-MaL called GP-tSNE was proposed that is specifically designed for producing interpretable visualisations of data. GP-tSNE showed strong evidence for the ability to use EC for feature manipulation in problems previously not tackled by EC methods. Visualisation methods consistently struggle to explain how produced visualisation were formed from the original dataset: GP-tSNE provided clear insight in this regard, especially when a series of increasingly complex visualisations were examined along the approximated Pareto front. This is a particularly important finding, as it is an advantage of GP-tSNE that is not possible in state-of-the-art manifold learning techniques.

Suitability of GP for Manifold Learning

The use of GP for MaL in Chapter 6 was motivated by the observation that manifold learning can be represented by learning a function that maps a high-dimensional dataset to a low-dimensional embedding. Such functions have previously not been proposed given the MaL community has focused on approaches that use gradient descent-style optimisation, which requires a differentiable cost function. GP, as an EC method, does not have this restriction: it can take advantage of non-differentiable function components such as conditional operators, which are much more powerful and flexible in representing manifolds with complex topologies.

The above motivation was convincingly validated through the results achieved by GP-MaL and GP-tSNE. While the raw performance of the GP-based methods may not have always matched those of the state-of-the-art methods, it is important to recognise that this is the **first** approach to using GP for MaL — the state-of-the-art MaL methods were created through many years of iterative improvement by a large and active research community. Continued research into the use of GP for MaL is likely to lead to further performance improvements. Furthermore, it was highlighted that creating a functional model that maps a manifold is strictly more difficult than directly creating the low-dimensional structure as an embedding (as t-SNE etc. do). In this regard, the GP MaL approaches proposed in this thesis show significant promise and novelty as they give both reasonable MaL performance as well as clear benefits for interpretability, which is often key in real-world use.

Understanding Data by Examining Increasingly Complex Models

While GP-tSNE showed a clear ability to produce interpretable visualisations, one of the most useful findings was the manner in which the approximated Pareto front produced by its EMO approach could be used to provide deeper insight. A single run of GP-tSNE produces many candidate visualisations at varying complexity (unlike t-SNE etc.). The

least complex of these gives a very coarse visualisation of the dataset, hopefully along the most defining axis of the manifold. While this visualisation is unlikely to fully show all the characteristics of the data, it provides one of the strongest patterns that is present. As we move along the front, models gradually get more complex, but still tend to maintain the earlier patterns found, while adding additional "detail" or aspects of the manifold at each step. This is seen in the GP trees through the addition of new features, and more complex combinations of features that encode feature interactions in the manifold. By adding a small amount of complexity at each step, it is much easier for humans to understand the increasingly complex trees as one only needs to focus on what has changed in the visualisation and tree compared to the previous slightly less complex model. This approach is unique to multi-objective ML methods, of which EC methods are seen as state-of-the-art. Indeed, this advantage of GP-tSNE is a strong and valuable contribution of this thesis. This approach is not necessarily specific to visualisation — other domains in which model complexity is critical could clearly benefit from such analysis.

7.2 Major Findings

While each of the four chapters in this thesis are self-contained investigations, there are a number of common conclusions reached, which provide particular insight into the use of EC for FM in unsupervised learning tasks:

1. The interpretability of models in unsupervised learning is even more critical than in supervised learning: unsupervised learning is part of data exploration where pre-existing knowledge (such as labels) is generally unavailable. Despite this, many techniques focus overly on raw performance with interpretability considered only as an afterthought, if at all. EC models that use FM are particularly suitable for addressing this, as they work by

simplifying a model's representation through smaller or more sophisticated feature spaces. This was especially true in the GP methods proposed in this thesis, whose interpretability was a key success. There is an obvious need for further efforts in this area.

2. The choice of fitness function in unsupervised learning is inherently more critical than in supervised learning. Unlike tasks such as classification and regression, which have ground truths, there is often no clear fundamental basis on which to model a fitness function. This may be further complicated when FM is used, such as when the number of features selected interact in a complex manner with the cluster quality measure (Chapter 3). In Chapters 4 to 6, two different approaches to fitness functions for the same (or similar) problems were proposed, with different benefits and limitations. There is a clear need for careful consideration and evaluation of fitness functions used in unsupervised domains.

3. Multi-tree GP approaches can be used to improve performance and interpretability (Chapters 4 to 6). It was discovered that GP is very well suited to producing individuals that exhibit heterogeneous behaviour across different trees, which is of particular benefit in complex tasks, which benefit from such niching-style approaches. Multi-tree clustering (Chapter 4) allowed different clusters to be treated in diverse ways; multi-tree redundant feature creation (Chapter 5) allowed multiple distinct features to be created; and GP-based MaL (Chapter 6) clearly necessitated a multi-tree approach to allow separate trees to encode discrete components of the manifold in a cohesive manner. Single-tree GP has historically been favoured in GP literature, but difficult unsupervised learning problems clearly benefit from a multi-tree approach.

4. The use of FM for unsupervised learning problems very often introduces two or more conflicting objectives. At the very least, complexity (# features or model size) should be traded-off against

performance. Often there are three or more objectives, such as when k is unknown in clustering or when the created redundant features should be distinct as well as redundant with the source feature. GP-tSNE (Chapter 6) clearly showcased the benefits of an EMO approach in this regard, as it allowed for a more elegant and useful balance between complexity and performance. Many unsupervised learning problems utilising FM, including those in this thesis, would likely benefit from an EMO approach — although, the increased computational cost of this would need to be carefully considered.

7.3 Future Work

The overall conclusions of this thesis (Section 7.2) provide clear direction for general work utilising EC for FM in unsupervised learning. This section will specifically highlight concrete areas of future work for each of the four contributions of this thesis in turn.

7.3.1 Particle Swarm Optimisation for Simultaneous Feature Selection and Clustering

Chapter 3 highlighted a number of opportunities for further improving PSO algorithms for simultaneous FS and clustering. In particular, an EMO approach could greatly help balance the three overall competing objectives of FS, cluster quality, and the number of clusters found. A clear interaction between these competing objectives necessitates finding a good balance that clearly provides understandable and representative clusters for a variety of data. Indeed, measuring clustering quality is in and of itself a multi-objective task due to the need to balance cluster compactness, separability, and connectedness. Subspace clustering also naturally benefits from FS, given the need to find clusters in different subspaces (i.e. feature subsets) of the data. Extending the approaches in this work to subspace clustering could be useful.

7.3.2 GP for Feature Construction in Clustering

Both the proposed approaches to using GP for FC in clustering (Chapter 4) were relatively unprecedented, leaving many possible directions for future research. These are discussed in turn for each approach below.

GP for Wrapper-Based FC in Clustering

As GP has seen little use in wrapper-based FC for clustering, there are many promising future research areas that could be explored. The representations and fitness functions proposed could be further improved, and many other representations and fitness functions are possible. For example, the vector approach could be adapted to directly encourage shorter constructed feature vectors to be produced thereby producing more powerful constructed features, by using a mechanism such as parsimony pressure. This would also significantly improve the interpretability of the constructed features. The multi-tree approach would be improved if the number of trees could be determined automatically — for example, using a heuristic based on K (a higher number of clusters should generally mean more constructed features are required). It may also be worthwhile to investigate using other clustering algorithms besides k-means; while in theory it is possible for GP to produce optimal constructed features that k-means can use to produce perfect partitions, other clustering algorithms may be more powerful and perform well with a wider range of constructed features. The methods we proposed were all designed to work when K was pre-defined, as k-means requires K to be known. This is somewhat inflexible, and extending these methods to automatically determine K would be beneficial.

GP for Evolving Similarity Functions

While the investigation in this thesis was focused on graph-based clustering, due to its ability to model a range of cluster shapes, we also hope our proposed approaches can be applied to nearly any clustering method that uses a similarity function to perform clustering. By replacing the graph-based clustering approach with another given clustering algorithm, the evolved similarity functions will need to be optimised to work with that algorithm instead. We would also like to test our proposed approaches on real-world data that has "gold standard" labels, but all real-world datasets we have found provide **class** labels only, which are not suitable for measuring cluster quality. This thesis focused on using a scalar fitness function so as to constrain the scope of this work, allowing us to directly evaluate the quality of the proposed GP representation. In future work, we would like to extend our proposed fitness function by using an evolutionary multi-objective optimisation (EMO) approach — the three key measures of cluster quality (compactness, separability, connectedness) partially conflict with each other, and so using an EMO approach may allow better and more varied solutions to be generated. Initial experiments showed that GPGC had promise for subspace clustering, but that better performance could likely be achieved in the future by developing a new fitness function and designing new genetic operators to be specific to subspace clustering tasks. There is also scope for refining the GP program design used: the terminals and functions could be further tailored to the clustering domain through the use of other feature comparison operators.

7.3.3 Generating Benchmark Feature Selection Datasets with GP

The use of GP for redundant feature creation (Chapter 5) showed significant potential for this new research area. Two promising areas of future work were identified.

Firstly, further investigation is needed into ways to make challenging redundant features that decrease FS algorithms' performance. One potential approach is the use of adversarial learning, where the performance of FS algorithm/s is directly used to evaluate the quality of candidate created features. Given the complex relationship between multivariate feature redundancies and the class label, an adversarial approach would allow a much more direct and stable optimisation to be performed. By creating features that are both meaningfully redundant as well as difficult for a variety of FS algorithms, useful benchmark FS datasets will be able to be developed. However, this would likely mean the redundant features created are no longer feature selection algorithm-agnostic, and so their use for general FS benchmarking may be limited.

The second direction is to take advantage of the finding that unsupervised redundant feature construction can actually improve classification performance through the creation of features that contain high levels of information about the class labels. Using MI (or similar) techniques could produce useful methods for performing FC as an (unsupervised) pre-processing step without the risk of feature bias, potentially improving supervised learning performance. This is a particularly unexpected and interesting discovery that warrants in-depth future investigation.

7.3.4 Interpretable Manifold Learning using GP

This thesis provided strong evidence for the development of interpretable manifold learning using GP in Chapter 6. Given that GP-MaL and GP-tSNE are the first methods to pursue this research direction, there are a range of possible directions of future work.

Future Work: GP-MaL

GP-MaL was the first general GP manifold learning technique, and could benefit from further iterative improvements to the fitness function and

representations used. GP-MaL is (purposefully) quite flexible, and could easily be extended further with other function sets and fitness functions (that do not have to be differentiable!). Inspiration from GP-tSNE could also be used by exploring a multi-objective approach that balances the often-conflicting objectives of maintaining global and local structure of a manifold. It is also clear that techniques to encourage simpler/more concise trees would further improve the usefulness of GP-MaL. For example, an EMO approach such as the one used in GP-tSNE could also be easily applied to GP-MaL to better encourage the creation of interpretable models.

Future Work: GP-tSNE

As GP-tSNE is the first multi-objective GP approach to balance unsupervised visualisation and model complexity, there is a clear need for continued future research. While the quality of visualisations were encouraging, on more complex datasets they failed to compare to those produced by t-SNE. This is not unexpected given t-SNE does not produce a functional mapping, but rather an embedding of the data, but further improvement to GP-tSNE's visualisations would make it significantly more useful in practice. The measure of visualisation quality used in this work was based on the one used by t-SNE given it is the state-of-the-art measure. However, the design of t-SNE was constrained by the need for a differentiable cost function; as an EC-based method, GP-tSNE need not have this constraint on its objective function — there is likely to be better measures of visualisation quality that could be used in an EC context.

Further study is also needed to determine which functions and terminals contribute most effectively to performance: we found that simple arithmetic operators were exclusively used in simple models. Designing more complex (but understandable) functions could allow more concise, powerful trees. It was also found that the more unique features used by a GP tree, the more difficult it was to understand; some sort of evolutionary pressure towards trees using a small set of (cohesive)

features may improve this.

Our analysis on the Dermatology dataset showed that it was possible for the location of two trees in an individual to be swapped without any penalty, but this makes the sequential visualisations harder to compare. Preventing crossover between trees in different positions in an individual would resolve this, but may restrict the evolutionary process from sharing useful sub-trees across the x and y-axes. Finally, it may be fruitful to investigate how to further improve the Pareto front analysis techniques used in this work for non-expert users such as through software graphical interfaces.

7.3.5 Final Thoughts

This thesis represents the first substantive investigation into the use of EC-based feature manipulation for unsupervised learning problems. Indeed, a common theme throughout this work was the dearth of literature that even tangentially considers this research direction. Given the promising results obtained and interesting patterns discovered in this work, there is clear motivation for wider research efforts in this area. Unsupervised learning remains a research frontier with many open questions and unaddressed problems. Evolutionary computation, and in particular EC-based feature manipulation, has untapped potential to give state-of-the-art and interpretable results in unsupervised learning.

Bibliography

[1] AGGARWAL, C. C., HINNEBURG, A., AND KEIM, D. A. On the surprising behavior of distance metrics in high dimensional spaces. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.* (2001), pp. 420–434.

[2] AGGARWAL, C. C., PROCOPIUC, C. M., WOLF, J. L., YU, P. S., AND PARK, J. S. Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (1999), pp. 61–72.

[3] AGGARWAL, C. C., AND REDDY, C. K., Eds. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[4] AHMED, S., ZHANG, M., PENG, L., AND XUE, B. Multiple feature construction for effective biomarker identification and classification using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada.* (2014), ACM, pp. 249–256.

[5] AHN, C. W., OH, S., AND OH, M. A genetic programming approach to data clustering. In *Proceedings of the International Conference on Multimedia, Computer Graphics and Broadcasting (MulGraB), Part II* (2011), pp. 123–132.

[6] AL-SAHAF, H., AL-SAHAF, A., XUE, B., JOHNSTON, M., AND ZHANG, M. Automatically evolving rotation-invariant texture

image descriptors by genetic programming. *IEEE Trans. Evolutionary Computation 21*, 1 (2017), 83–101.

[7] AL-SAHAF, H., BI, Y., CHEN, Q., LENSEN, A., MEI, Y., SUN, Y., TRAN, B., XUE, B., AND ZHANG, M. A survey on evolutionary machine learning. *Journal of the Royal Society of New Zealand* (2019). To be published.

[8] ALELYANI, S., TANG, J., AND LIU, H. Feature selection for clustering: A review. In *Data Clustering: Algorithms and Applications*. CRC Press, 2013, pp. 29–60.

[9] ALPAYDIN, E. *Introduction to Machine Learning*. MIT Press, 2014.

[10] AMALDI, E., AND KANN, V. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theor. Comput. Sci. 209*, 1-2 (1998), 237–260.

[11] ANKERST, M., BREUNIG, M. M., KRIEGEL, H., AND SANDER, J. OPTICS: ordering points to identify the clustering structure. In *Proceedings of the International Conference on Management of Data* (1999), pp. 49–60.

[12] ARTHUR, D., AND VASSILVITSKII, S. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2007), pp. 1027–1035.

[13] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[14] BENGIO, Y., COURVILLE, A. C., AND VINCENT, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell. 35*, 8 (2013), 1798–1828.

[15] BEYER, K. S., GOLDSTEIN, J., RAMAKRISHNAN, R., AND SHAFT, U. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT)* (1999), pp. 217–235.

[16] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Trans. Evolutionary Computation 17*, 3 (2013), 368–386.

[17] BI, J., BENNETT, K. P., EMBRECHTS, M. J., BRENEMAN, C. M., AND SONG, M. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research 3* (2003), 1229–1243.

[18] BLEULER, S., BRACK, M., THIELE, L., AND ZITZLER, E. Multiobjective genetic programming: Reducing bloat using spea2. In *Proceedings of the Congress on Evolutionary Computation (CEC)* (2001), IEEE, pp. 536–543.

[19] BLICKLE, T. Evolving compact solutions in genetic programming: A case study. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature - PPSN IV* (1996), vol. 1141 of *Lecture Notes in Computer Science*, Springer, pp. 564–573.

[20] BORIC, N., AND ESTÉVEZ, P. A. Genetic programming-based clustering using an information theoretic fitness measure. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)* (2007), pp. 31–38.

[21] BRADLEY, P. S., FAYYAD, U. M., AND REINA, C. A. Scaling em (expectation maximization) clustering to large databases. Tech. Rep. MSR-TR-98-35, Microsoft, November 1998.

[22] BRADLEY, P. S., AND MANGASARIAN, O. L. Feature selection via concave minimization and support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998* (1998), pp. 82–90.

[23] CANO, A., VENTURA, S., AND CIOS, K. J. Multi-objective genetic programming for feature extraction and data visualization. *Soft Comput. 21*, 8 (2017), 2069–2089.

[24] CHIANG, M. M., AND MIRKIN, B. G. Intelligent choice of the number of clusters in *K*-means clustering: An experimental study with different cluster spreads. *J. Classification 27*, 1 (2010), 3–40.

[25] COELHO, A. L. V., FERNANDES, E., AND FACELI, K. Multi-objective design of hierarchical consensus functions for clustering ensembles via genetic programming. *Decision Support Systems 51*, 4 (2011), 794–809.

[26] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[27] CURA, T. A particle swarm optimization approach to clustering. *Expert Syst. Appl. 39*, 1 (2012), 1582–1588.

[28] DAVIES, D. L., AND BOULDIN, D. W. A cluster separation measure. *IEEE Trans. Pattern Anal. Mach. Intell. 1*, 2 (1979), 224–227.

[29] DEB, K., AGRAWAL, S., PRATAP, A., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation 6*, 2 (2002), 182–197.

[30] DENG, H., AND RUNGER, G. C. Feature selection via regularized trees. In *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012* (2012), pp. 1–8.

[31] DHEERU, D., AND KARRA TANISKIDOU, E. UCI machine learning repository, 2017.

[32] DUNN, J. C. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics 3*, 3 (1973), 32–57.

[33] DY, J. G., AND BRODLEY, C. E. Feature selection for unsupervised learning. *Journal of Machine Learning Research 5* (2004), 845–889.

[34] ECLIPSE DEEPLEARNING4J DEVELOPMENT TEAM. Deeplearning4j: Open-source distributed deep learning for the JVM. `http:`

//deeplearning4j.org, 2019. Apache Software Foundation License 2.0.

[35] EIBEN, A. E., AND SMITH, J. E. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2015.

[36] ESMIN, A. A. A., COELHO, R. A., AND MATWIN, S. A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data. *Artif. Intell. Rev. 44*, 1 (2015), 23–45.

[37] ESPEJO, P. G., VENTURA, S., AND HERRERA, F. A survey on the application of genetic programming to classification. *IEEE Trans. Systems, Man, and Cybernetics, Part C 40*, 2 (2010), 121–144.

[38] ESTER, M., KRIEGEL, H., SANDER, J., AND XU, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA* (1996), pp. 226–231.

[39] FALCO, I. D., TARANTINO, E., CIOPPA, A. D., AND GAGLIARDI, F. A novel grammar-based genetic programming approach to clustering. In *Proceedings of the ACM Symposium on Applied Computing (SAC)* (2005), pp. 928–932.

[40] FALKENAUER, E. *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc., 1998.

[41] FAYYAD, U., PIATETSKY-SHAPIRO, G., AND SMYTH, P. From data mining to knowledge discovery in databases. *AI Magazine 17*, 3 (1996), 37–54.

[42] FRANÇOIS, D., WERTZ, V., AND VERLEYSEN, M. The concentration of fractional distances. *IEEE Trans. Knowl. Data Eng. 19*, 7 (2007), 873–886.

[43] FRÄNTI, P., AND SIERANOJA, S. K-means properties on six clustering benchmark datasets. *Appl. Intell. 48*, 12 (2018), 4743–4759. `http://cs.uef.fi/sipu/datasets/` (Last accessed: 27/09/19).

[44] FREITAS, A. A. *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science & Business Media, 2013.

[45] FRIEDMAN, H. P., AND RUBIN, J. On some invariant criteria for grouping data. *Journal of the American Statistical Association 62*, 320 (1967), 1159–1178.

[46] GARCÍA, A. J., AND GÓMEZ-FLORES, W. Automatic clustering using nature-inspired metaheuristics: A survey. *Appl. Soft Comput. 41* (2016), 192–213.

[47] GARCÍA-PEDRAJAS, N., DE HARO-GARCÍA, A., AND PÉREZ-RODRÍGUEZ, J. A scalable memetic algorithm for simultaneous instance and feature selection. *Evolutionary Computation 22*, 1 (2014), 1–45.

[48] GIROLAMI, M. A. Mercer kernel-based clustering in feature space. *IEEE Trans. Neural Networks 13*, 3 (2002), 780–784.

[49] GORBAN, A. N., KÉGL, B., WUNSCH, D. C., ZINOVYEV, A. Y., ET AL. *Principal manifolds for data visualization and dimension reduction*, vol. 58. Springer, 2008.

[50] GRAHAM, R. L., KNUTH, D. E., AND PATASHNIK, O. *Concrete mathematics - a foundation for computer science*. Addison-Wesley, 1989.

[51] GUO, H., AND NANDI, A. K. Breast cancer diagnosis using genetic programming generated feature. *Pattern Recognition 39*, 5 (2006), 980–987.

[52] GUYON, I. Design of experiments of the nips 2003 variable selection benchmark. In *NIPS 2003 workshop on feature extraction and feature selection* (2003).

[53] GUYON, I., AND ELISSEEFF, A. An introduction to variable and feature selection. *Journal of Machine Learning Research 3* (2003), 1157–1182.

[54] GUYON, I., GUNN, S. R., BEN-HUR, A., AND DROR, G. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]* (2004), pp. 545–552.

[55] HALL, M. A., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The WEKA data mining software: an update. *SIGKDD Explorations 11*, 1 (2009), 10–18.

[56] HANDL, J., AND KNOWLES, J. D. An evolutionary approach to multiobjective clustering. *IEEE Trans. Evolutionary Computation 11*, 1 (2007), 56–76.

[57] HANSEN, N., MÜLLER, S. D., AND KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation 11*, 1 (2003), 1–18.

[58] HART, E., SIM, K., GARDINER, B., AND KAMIMURA, K. A hybrid method for feature construction and selection to improve wind-damage prediction in the forestry sector. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017* (2017), pp. 1121–1128.

[59] HARTUV, E., AND SHAMIR, R. A clustering algorithm based on graph connectivity. *Inf. Process. Lett. 76*, 4-6 (2000), 175–181.

[60] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009.

[61] HAYNES, T., AND SEN, S. Crossover operators for evolving a team. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (CA, USA, 1997), J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds., Morgan Kaufmann, pp. 162–167.

[62] HINTON, G. E., AND ROWEIS, S. T. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]* (2002), pp. 833–840.

[63] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[64] HRUSCHKA, E. R., CAMPELLO, R. J. G. B., FREITAS, A. A., AND DE CARVALHO, A. C. P. L. F. A survey of evolutionary algorithms for clustering. *IEEE Trans. Systems, Man, and Cybernetics, Part C 39*, 2 (2009), 133–155.

[65] HSU, W., ZHANG, Y., AND GLASS, J. R. Unsupervised learning of disentangled and interpretable representations from sequential data. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS) 30* (2017), pp. 1876–1887.

[66] ICKE, I., AND ROSENBERG, A. Multi-objective genetic programming for visual analytics. In *Proceedings of the European Conference on Genetic Programming (EuroGP)* (2011), pp. 322–334.

[67] INABA, M., KATOH, N., AND IMAI, H. Applications of weighted voronoi diagrams and randomization to variance-based *k*-clustering (extended abstract). In *Proceedings of the Tenth Annual Symposium on Computational Geometry* (1994), pp. 332–339.

[68] J. A. HARTIGAN, M. A. W. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*, 1 (1979), 100–108.

[69] JAIN, A. K. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters 31*, 8 (2010), 651–666.

[70] JAIN, A. K., DUIN, R. P. W., AND MAO, J. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 1 (2000), 4–37.

[71] JAVANI, M., FAEZ, K., AND AGHLMANDI, D. Clustering and feature selection via PSO algorithm. In *International Symposium on Artificial Intelligence and Signal Processing (AISP)* (2011), IEEE, pp. 71–76.

[72] JAYNES, E. T. Information theory and statistical mechanics. *Physical review 106*, 4 (1957), 620.

[73] JIN, Y. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation 1*, 2 (2011), 61–70.

[74] JOLLIFFE, I. T. Principal component analysis. In *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 1094–1096.

[75] JORDAN, M. I., AND MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science 349*, 6245 (2015), 255–260.

[76] KAUFMAN, L., AND ROUSSEEUW, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[77] KENNEDY, J. Swarm intelligence. In *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.

[78] KENNEDY, J., AND EBERHART, R. C. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[79] KOHAVI, R., AND JOHN, G. H. Wrappers for feature subset selection. *Artif. Intell. 97*, 1-2 (1997), 273–324.

[80] KOHONEN, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43*, 1 (Jan 1982), 59–69.

[81] KOHONEN, T. *Self-organization and associative memory*, vol. 8. Springer Science & Business Media, 2012.

[82] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[83] KRASKOV, A., STÖGBAUER, H., AND GRASSBERGER, P. Estimating mutual information. *Physical review E 69*, 6 (2004), 066138.

[84] KRUSKAL, J. B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika 29*, 1 (Mar 1964), 1–27.

[85] KUO, R. J., SYU, Y. J., CHEN, Z., AND TIEN, F. Integration of particle swarm optimization and genetic algorithm for dynamic clustering. *Inf. Sci. 195* (2012), 124–140.

[86] LANE, M. C., XUE, B., LIU, I., AND ZHANG, M. Particle swarm optimisation and statistical clustering for feature selection. In *AI 2013: Advances in Artificial Intelligence*, vol. 8272 of *Lecture Notes in Computer Science*. Springer International Publishing, 2013, pp. 214–220.

[87] LANE, M. C., XUE, B., LIU, I., AND ZHANG, M. Gaussian based particle swarm optimisation and statistical clustering for feature selection. In *Evolutionary Computation in Combinatorial Optimisation*, vol. 8600 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 133–144.

[88] LEE, J. A., AND VERLEYSEN, M. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[89] LENSEN, A., XUE, B., AND ZHANG, M. Particle swarm optimisation representations for simultaneous clustering and feature selection. In *Proceedings of the Symposium Series on Computational Intelligence* (2016), IEEE, pp. 1–8.

[90] LENSEN, A., XUE, B., AND ZHANG, M. GPGC: genetic programming for automatic clustering using a flexible non-hyper-spherical graph-based approach. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO.* (2017), ACM, pp. 449–456.

[91] LENSEN, A., XUE, B., AND ZHANG, M. Improving *k*-means clustering with genetic programming for feature construction. In *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings* (2017), P. A. N. Bosman, Ed., ACM, pp. 237–238.

[92] LENSEN, A., XUE, B., AND ZHANG, M. New representations in genetic programming for feature construction in k-means clustering. In *Proceedings of the 11th International Conference on Simulated Evolution and Learning (SEAL)* (2017), vol. 10593 of *Lecture Notes in Computer Science*, Springer, pp. 543–555.

[93] LENSEN, A., XUE, B., AND ZHANG, M. Using particle swarm optimisation and the silhouette metric to estimate the number of clusters, select features, and perform clustering. In *Proceedings of the 20th European Conference on the Applications of Evolutionary Computation (EvoApplications), Part I* (2017), vol. 10199 of *Lecture Notes in Computer Science*, Springer, pp. 538–554.

[94] LENSEN, A., XUE, B., AND ZHANG, M. Automatically evolving difficult benchmark feature selection datasets with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO* (2018), ACM, pp. 458–465.

[95] LENSEN, A., XUE, B., AND ZHANG, M. Generating redundant features with unsupervised multi-tree genetic programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP)* (2018), vol. 10781 of *Lecture Notes in Computer Science*, Springer, pp. 84–100.

[96] LENSEN, A., XUE, B., AND ZHANG, M. Can genetic programming do manifold learning too? In *Proceedings of the European Conference on Genetic Programming (EuroGP)* (2019), vol. 11451 of *Lecture Notes in Computer Science*, Springer, pp. 114–130. Awarded best paper in EuroGP.

[97] LIN, Y., AND BHANU, B. Evolutionary feature synthesis for object recognition. *IEEE Trans. Systems, Man, and Cybernetics, Part C 35*, 2 (2005), 156–171.

[98] LIU, H., AND MOTODA, H. *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media, 1998.

[99] LIU, H., AND MOTODA, H. *Feature selection for knowledge discovery and data mining*, vol. 454. Springer Science & Business Media, 2012.

[100] LIU, H., MOTODA, H., SETIONO, R., AND ZHAO, Z. Feature selection: An ever evolving frontier in data mining. In *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining* (2010), pp. 4–13.

[101] LIU, H., AND YU, L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. Knowl. Data Eng. 17*, 4 (2005), 491–502.

[102] LIU, H., AND ZHAO, Z. Manipulating data and dimension reduction methods: Feature selection. In *Encyclopedia of Complexity and Systems Science*. Springer, 2009, pp. 5348–5359.

[103] LIZIER, J. T. JIDT: an information-theoretic toolkit for studying the dynamics of complex systems. *Front. Robotics and AI 1* (2014), 11.

[104] LORENA, L. A. N., AND FURTADO, J. C. Constructive genetic algorithm for clustering problems. *Evolutionary Computation 9*, 3 (2001), 309–328.

[105] LOVEARD, T., AND CIESIELSKI, V. Representing classification problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation* (2001), vol. 2, IEEE, pp. 1070–1077.

[106] LUKE, S., AND PANAIT, L. Lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (2002), pp. 829–836.

[107] LUKE, S., AND PANAIT, L. A comparison of bloat control methods for genetic programming. *Evolutionary Computation (Journal) 14*, 3 (2006), 309–344.

[108] MARILL, T., AND GREEN, D. M. On the effectiveness of receptors in recognition systems. *IEEE Trans. Information Theory 9*, 1 (1963), 11–17.

[109] MARINAKIS, Y., MARINAKI, M., AND MATSATSINIS, N. F. A hybrid particle swarm optimization algorithm for clustering analysis. In *Proceedings of the 9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)* (2007), pp. 241–250.

[110] MARINAKIS, Y., MARINAKI, M., AND MATSATSINIS, N. F. A hybrid clustering algorithm based on multi-swarm constriction PSO and GRASP. In *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)* (2008), pp. 186–195.

[111] MCINNES, L., AND HEALY, J. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR abs/1802.03426* (2018).

[112] MEILĂ, M. Comparing clusterings—an information based distance. *Journal of multivariate analysis 98*, 5 (2007), 873–895.

[113] MENÉNDEZ, H. D., BARRERO, D. F., AND CAMACHO, D. A genetic graph-based approach for partitional clustering. *International Journal of Neural Systems 24*, 3 (2014), 19.

[114] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M. A., FIDJELAND, A., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[115] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012.

[116] MÜLLER, E., GÜNNEMANN, S., ASSENT, I., AND SEIDL, T. Evaluating clustering in subspace projections of high dimensional data. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)* (2009), pp. 1270–1281.

[117] MUNI, D. P., PAL, N. R., AND DAS, J. Genetic programming for simultaneous feature selection and classifier design. *IEEE Trans. Systems, Man, and Cybernetics, Part B 36*, 1 (2006), 106–117.

[118] NANDA, S. J., AND PANDA, G. A survey on nature inspired metaheuristic algorithms for partitional clustering. *Swarm and Evolutionary Computation 16* (2014), 1–18.

[119] NAREDO, E., AND TRUJILLO, L. Searching for novel clustering programs. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013* (2013), pp. 1093–1100.

[120] NESHATIAN, K., ZHANG, M., AND ANDREAE, P. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Trans. Evolutionary Computation 16*, 5 (2012), 645–661.

[121] NGUYEN, H. B., XUE, B., LIU, I., ANDREAE, P., AND ZHANG, M. Gaussian transformation based representation in particle swarm

optimisation for feature selection. In *Applications of Evolutionary Computation*, vol. 9028 of *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 541–553.

[122] NGUYEN, H. B., XUE, B., LIU, I., AND ZHANG, M. PSO and statistical clustering for feature selection: A new representation. In *Simulated Evolution and Learning*, vol. 8886 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 569–581.

[123] NGUYEN, S., ZHANG, M., ALAHAKOON, D., AND TAN, K. C. Visualizing the evolution of computer programs for genetic programming [research frontier]. *IEEE Computational Intelligence Magazine 13*, 4 (Nov 2018), 77–94.

[124] NGUYEN, X. V., EPPS, J., AND BAILEY, J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)* (2009), pp. 1073–1080.

[125] O'NEILL, D., LENSEN, A., XUE, B., AND ZHANG, M. Particle swarm optimisation for feature selection and weighting in high-dimensional clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC* (2018), IEEE, pp. 1–8.

[126] PAL, N. R., AND BEZDEK, J. C. On cluster validity for the fuzzy c-means model. *IEEE Trans. Fuzzy Systems 3*, 3 (1995), 370–379.

[127] PARK, H., AND JUN, C. A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl. 36*, 2 (2009), 3336–3341.

[128] PARSONS, L., HAQUE, E., AND LIU, H. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations 6*, 1 (2004), 90–105.

[129] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER,

P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[130] PEIGNIER, S., RIGOTTI, C., AND BESLON, G. Subspace clustering using evolvable genome structure. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO* (2015), pp. 575–582.

[131] PENG, H., LONG, F., AND DING, C. H. Q. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 8 (2005), 1226–1238.

[132] PENN, B. S. Using self-organizing maps to visualize high-dimensional data. *Computers & Geosciences 31*, 5 (2005), 531–544.

[133] PHAM, D. T., DIMOV, S. S., AND NGUYEN, C. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 219*, 1 (2005), 103–119.

[134] PICAROUGNE, F., AZZAG, H., VENTURINI, G., AND GUINOT, C. A new approach of data clustering using a flock of agents. *Evolutionary Computation 15*, 3 (2007), 345–367.

[135] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. lulu.com, 2008. (Last Accessed: 27/09/19).

[136] POLI, R., AND MCPHEE, N. F. Parsimony pressure made easy: Solving the problem of bloat in GP. In *Theory and Principled Methods for the Design of Metaheuristics*. Springer, 2014, pp. 181–204.

[137] PROCOPIUC, C. M., JONES, M., AGARWAL, P. K., AND MURALI, T. M. A monte carlo algorithm for fast projective clustering. In *Proceedings of theACM SIGMOD International Conference on Management of Data* (2002), pp. 418–427.

[138] PUDIL, P., NOVOVICOVÁ, J., AND KITTLER, J. Floating search methods in feature selection. *Pattern Recognition Letters 15*, 10 (1994), 1119–1125.

[139] RAKOTOMAMONJY, A. Variable selection using SVM-based criteria. *Journal of Machine Learning Research 3* (2003), 1357–1370.

[140] RAND, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*, 336 (1971), 846–850.

[141] RODRIGUEZ-COAYAHUITL, L., MORALES-REYES, A., AND ESCALANTE, H. J. Structurally layered representation learning: Towards deep learning through genetic programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP)* (2018), pp. 271–288.

[142] ROKACH, L., AND MAIMON, O. Clustering methods. In *The Data Mining and Knowledge Discovery Handbook.* Springer, 2005, pp. 321–352.

[143] ROUSSEEUW, P. J. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics 20* (1987), 53 – 65.

[144] ROWEIS, S. T., AND SAUL, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science 290*, 5500 (2000), 2323–2326.

[145] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[146] SCHAEFFER, S. E. Graph clustering. *Computer Science Review 1*, 1 (2007), 27–64.

[147] SCHONLAU, M. Visualizing non-hierarchical and hierarchical cluster analyses with clustergrams. *Computational Statistics 19*, 1 (2004), 95–111.

[148] SHAND, C., ALLMENDINGER, R., HANDL, J., WEBB, A. M., AND KEANE, J. Evolving controllably difficult datasets for clustering. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO)* (2019), pp. 463–471.

[149] SHENG, W., CHEN, S., SHENG, M., XIAO, G., MAO, J., AND ZHENG, Y. Adaptive multisubpopulation competition and multiniche crowding-based memetic algorithm for automatic data clustering. *IEEE Trans. Evolutionary Computation 20*, 6 (2016), 838–858.

[150] SHENG, W., LIU, X., AND FAIRHURST, M. C. A niching memetic algorithm for simultaneous clustering and feature selection. *IEEE Trans. Knowl. Data Eng. 20*, 7 (2008), 868–879.

[151] SILVER, D., HUBERT, T., SCHRITTWIESER, J., ANTONOGLOU, I., LAI, M., GUEZ, A., LANCTOT, M., SIFRE, L., KUMARAN, D., GRAEPEL, T., LILLICRAP, T., SIMONYAN, K., AND HASSABIS, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[152] SIMONITE, T. The missing link of artificial intelligence. *MIT Technology Review* (2016). `https://www.technologyreview.com/s/600819/the-missing-link-of-artificial-intelligence/` (Last Accessed: 27/09/19).

[153] SOKAL, R. R., AND MICHENER, C. D. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin 28* (1958), 1409–1438.

[154] SONDHI, P. Feature construction methods: A survey. Tech. rep., Univeristy of Illinois at Urbana Champaign, Urbana, Illinois, USA, 2009.

[155] SOULE, T., AND FOSTER, J. A. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation 6*, 4 (1998), 293–309.

[156] STATNIKOV, A., HARDIN, D., AND ALIFERIS, C. Using SVM weight-based methods to identify causally relevant and non-causally relevant variables. In *Proceeding of the Neural Information Processing Systems (NIPS) Workshop on Causality and Feature Selection* (2006), p. 11.

[157] SUN, Y., XUE, B., ZHANG, M., AND YEN, G. G. A particle swarm optimization-based flexible convolutional auto-encoder for image classification. *IEEE Trans. Neural Networks and Learning Systems 30*, 8 (2019), 2295–2309.

[158] SUN, Y., YEN, G. G., AND YI, Z. Evolving unsupervised deep neural networks for learning meaningful representations. *IEEE Trans. Evolutionary Computation 23*, 1 (2019), 89–103.

[159] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[160] TANG, J., ALELYANI, S., AND LIU, H. Feature selection for classification: A review. In *Data Classification: Algorithms and Applications*. CRC Press, 2014, pp. 37–64.

[161] TENENBAUM, J. B., SILVA, V. D., AND LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. *Science 290*, 5500 (2000), 2319–2323.

[162] THOMASON, R., AND SOULE, T. Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO)* (2007), pp. 1708–1715.

[163] TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological) 58*, 1 (1996), 267–288.

[164] TIBSHIRANI, R., WALTHER, G., AND HASTIE, T. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63*, 2 (2001), 411–423.

[165] TRAN, B., XUE, B., AND ZHANG, M. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing 8*, 1 (2016), 3–15.

[166] TSENG, L. Y., AND YANG, S. B. A genetic clustering algorithm for data with non-spherical-shape clusters. *Pattern Recognition 33*, 7 (2000), 1251–1259.

[167] VAHDAT, A., AND HEYWOOD, M. I. On evolutionary subspace clustering with symbiosis. *Evolutionary Intelligence 6*, 4 (2014), 229–256.

[168] VAN DEN BERGH, F. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, 2006.

[169] VAN DER MAATEN, L. Learning a parametric embedding by preserving local structure. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS* (2009), pp. 384–391.

[170] VAN DER MAATEN, L. Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research 15*, 1 (2014), 3221–3245.

[171] VAN DER MAATEN, L., AND HINTON, G. E. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research 9* (2008), 2579–2605.

[172] VAN DONGEN, S. M. *Graph clustering by flow simulation*. PhD thesis, University of Utrecht, May 2000.

[173] VINH, N. X., EPPS, J., AND BAILEY, J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research 11* (2010), 2837–2854.

[174] VON LUXBURG, U. A tutorial on spectral clustering. *Statistics and Computing 17*, 4 (2007), 395–416.

[175] VON LUXBURG, U., WILLIAMSON, R. C., AND GUYON, I. Clustering: Science or Art? In *Proceedings of the Unsupervised and Transfer Learning Workshop held at ICML 2011* (2012), pp. 65–80.

[176] WAGNER, M., AND NEUMANN, F. Parsimony pressure versus multi-objective optimization for variable length representations. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN) XII* (2012), pp. 133–142.

[177] WHITNEY, A. W. A direct method of nonparametric measurement selection. *IEEE Trans. Computers 20*, 9 (1971), 1100–1103.

[178] XU, R., AND II, D. C. W. Survey of clustering algorithms. *IEEE Trans. Neural Networks 16*, 3 (2005), 645–678.

[179] XUE, B., ZHANG, M., BROWNE, W. N., AND YAO, X. A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evolutionary Computation 20*, 4 (2016), 606–626.

[180] YANG, H., AND MOODY, J. Feature selection based on joint mutual information. In *Proceedings of Computational Intelligence Methods and Applications (CIMA), New York, USA* (1999), pp. 22–25.

[181] ZENG, Z., ZHANG, H., ZHANG, R., AND YIN, C. A novel feature selection method considering feature interaction. *Pattern Recognition 48*, 8 (2015), 2656–2666.

[182] ZHANG, C., LIU, C., ZHANG, X., AND ALMPANIDIS, G. An up-to-date comparison of state-of-the-art classification algorithms. *Expert Systems with Applications 82* (2017), 128–150.

[183] ZHANG, Q., LEI, X., HUANG, X., AND ZHANG, A. An improved projection pursuit clustering model and its application based on quantum-behaved PSO. In *Proceedings of the Sixth International Conference on Natural Computation (ICNC)* (2010), pp. 2581–2585.

[184] ZHANG, Y., AND ZHANG, M. A multiple-output program tree structure in genetic programming. Tech. rep., Victoria University of Wellington, New Zealand, 2004.

[185] ZHOU, A., QU, B., LI, H., ZHAO, S., SUGANTHAN, P. N., AND ZHANG, Q. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation 1*, 1 (2011), 32–49.

[186] ZHU, J., ROSSET, S., HASTIE, T., AND TIBSHIRANI, R. 1-norm support vector machines. In *Advances in Neural Information Processing Systems (NIPS) 16* (2003), pp. 49–56.

[187] ZITZLER, E., LAUMANNS, M., AND THIELE, L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland, 2001.