Noname manuscript No.
(will be inserted by the editor)

Genetic Programming for Feature Construction and Selection in Classification on High-dimensional Data

Binh Tran*, Bing Xue* and Mengjie Zhang

Received: 211 July 2015 / Accepted: 7 December 2015

Abstract Classification on high-dimensional data with thousands to tens of thousands of dimensions is a challenging task due to the high dimensionality and the quality of the feature set. The problem can be addressed by using *feature selection* to choose only informative features or *feature construction* to create new high-level features. Genetic programming (GP) using a tree-based representation can be used for both feature construction and implicit feature selection. This work presents a comprehensive study to investigate the use of GP for feature construction and selection on high-dimensional classification problems. Different combinations of the constructed and/or selected features are tested and compared on seven high-dimensional gene expression problems, and different classification algorithms are used to evaluate their performance. The results show that the constructed and/or selected feature sets can significantly reduce the dimensionality and maintain or even increase the classification accuracy in most cases. The cases with overfitting occurred are analysed via the distribution of features. Further analysis is also performed to show why the constructed feature can achieve promising classification performance.

Keywords Genetic programming · Feature construction · Feature selection · Classification · High-dimensional data.

*Corresponding authors.

Authors
Evolutionary Computation Research Group,
Victoria University of Wellington,
PO Box 600, Wellington 6140, New Zealand
Tel.: +64-4-463-5233;
fax: +64-4-463-5045;
E-mail: binh.tran@ecs.vuw.ac.nz
E-mail: bing.xue@ecs.vuw.ac.nz
E-mail: mengjie.zhang@ecs.vuw.ac.nz

1 Introduction

Classification is a major supervised machine learning task that aims to classify an instance into its corresponding category [20,28,31]. A classification problem is typically described by a set of features and the class labels. The quality of the feature set is a key factor influencing the performance of a classification/learning algorithm [28]. Irrelevant and redundant features may negatively affect the classification accuracy, increase complexity of the learnt classifier and the running time.

Feature selection and feature construction are data preprocessing techniques used to enhance the quality of the feature space. Feature selection aims at selecting only useful features from the original feature set. Feature construction combines original features to obtain new high-level features that may provide better discrimination for the problem [22].

Three different types of feature selection and construction approaches have been proposed: wrapper, filter and embedded approaches [7]. While the classification performance of a learning algorithm is used as the evaluation criterion in wrapper methods, intrinsic characteristics of the data is used in filter methods. Wrapper methods thereby usually require a higher computation time, but the selected or constructed features usually have better performance than those selected or constructed by filter methods. Embedded methods simultaneously select or construct features and learn a classifier.

Evolutionary computation has been widely used for feature selection [8,23] as well as feature construction [26]. Among many evolutionary algorithms, genetic programming (GP) is a very flexible technique that can automatically evolve mathematical models without a predefined template such as linear or non-linear. GP allows

us to use complex representations such as trees with any kind of operators or functions to represent the model. These properties make GP an excellent choice for feature construction where the learnt model is a mathematical function used to generate new features based on the original ones. GP constructed features have been used to effectively augment the original feature space of the problem [17, 22].

GP is not only used to construct new high-level features, it is also used as a *feature selection* method [2, 18]. Since a GP tree does not use all the original features to construct the high-level features, the features that are used in leave nodes of the tree, i.e. terminal features, are useful/informative features. Feature selection can be achieved by using the terminal (original) features for classification [24]. Although many GP algorithms have been proposed for feature construction and feature selection, most studies have been applied on small datasets with about tens of features.

Recently, Ahmed et al proposed a GP-based feature construction method in [3] to construct multiple features by forming new features based on all possible subtrees in the best individual. The method was applied on mass spectrometry data with thousands to tens of thousands of features. A comparison of the constructed features with the selected features in the best individual was conducted. The results showed that the constructed feature achieved better performance on common classification algorithms. However, no investigation has been done on different combinations of GP constructed and/or selected features using this approach. In addition, in order to achieve the best classification performance with these GP generated feature sets, it is important to know which combination of selected and constructed features works better than others in general or for a specific learning algorithm.

Goals

In this study, we aim to investigate the potential of GP in feature construction and feature selection for classification on high-dimensional data. A feature construction embedded approach is used here, where a GP tree itself is a constructed high-level feature and also a classifier. Since each possible subtree also represents a constructed high-level feature, a GP tree implicitly construct multiple new high-level features [3]. Furthermore, feature selection can be achieved by using only the terminal (original) features for classification. Therefore, an embedded GP for feature construction approach produces new high-level features, a set of original useful features, and a classifier. In this work, we aim to investigate the following questions:

1. Whether GP can select informative features from thousands of original features.
2. Whether GP can construct features that improve the performance of common classification algorithms on gene expression data.
3. Which combinations of the constructed and/or selected features can work better for common learning algorithms.

2 Background

2.1 Genetic programming algorithm

GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. As a population-based evolutionary computation technique, GP typically follows these steps:

1. Generate an initial population of individuals (trees or programs) composing of primitive functions and terminals of the problem.
2. Iteratively perform the following sub-steps until a stopping criterion is met:
   (a) Evaluation: Each individual is executed and its fitness is calculated based on a predefined fitness function.
   (b) Selection: Select one or two individuals from the population with a probability based on fitness to participate in the evolution step.
   (c) Evolution: Create new individuals for the new population by applying the following genetic operators with specific probabilities:
      i. *Reproduction:* Copy the selected individuals to the new population.
      ii. *Crossover:* Create new offsprings by recombining randomly chosen parts from two selected programs.
      iii. *Mutation:* Create a new offspring program by randomly mutating a randomly chosen part of one selected program.
3. Return the program with the highest fitness as the best solution.

Applying GP to solving problem has to specify a terminal set, a function set, a fitness function, a stopping criterion and control parameters such as population size, crossover and mutation probabilities.

2.2 GP for feature construction and selection

Although application of GP for inducing classifiers usually implies a feature selection and construction process

[10], this section is restricted to studies that use GP explicitly for preprocessing purposes.

When using GP for feature selection or feature construction, the terminal set comprises of features chosen from the original features and/or random constants. The function set typically comprises of mathematical operators.

As feature selection is an intrinsic characteristic of GP, some studies have attempted to use GP for feature selection. Features appeared in the best GP individuals are used to form a feature subset [2]. Feature ranking or feature weighting is also achieved by counting the occurrence of features in the good individuals [25]. Although these methods obtained promising results, the most fruitful application of GP in preprocessing task is still feature construction thanks to its ability to combine original features in such a flexible way that can create better discriminating features.

GP has been proposed as filter or wrapper feature construction using single or multiple-tree representation to construct a single or multiple features.

2.2.1 GP for Single Feature Construction

GP has been proposed to construct a single new feature for a given problem. These methods used the single-tree GP representation. Each individual program in the population is represented by a single tree and forms a constructed feature. In [22], a filter GP-based single feature construction was proposed using four different measures in fitness evaluations including information gain, the gini index, a combination of information gain and gini index, and chi-square. The augmented feature sets were evaluated using three decision tree algorithms (C5, CHAID, CART) and a multilayer perceptron. Results on five datasets with 4 to 21 features showed that the constructed feature in general improved the performance of all classifiers without any bias toward the fitness measures used in feature construction process. A similar single-tree GP approach to [22] but using scattering between class as a fitness measure was proposed in [12]. Results on Breast cancer dataset with 30 features showed that the one constructed feature by GP outperformed two to five features extracted by principle component analysis and three other methods based on Fisher linear discriminant analysis. Although these methods have shown promises, applying them on high-dimensional data needs further investigation.

2.2.2 GP for Multiple Feature Construction

Unlike single-feature construction methods, multiple-feature construction methods employ different strategies in GP. A straight forward strategy is to use multi-tree GP in which a GP individual program comprises a number of trees, each corresponding to a constructed feature. In [16], each GP individual represents predefined numbers of constructed features and hidden features. Hidden features were used as an elitist repository which were updated from constructed features that had highest usage frequency in the decision tree learnt in the fitness function. A similar multi-tree GP feature construction method was proposed in [11] without using hidden features. Results of these methods showed that the proposed methods achieved competitive prediction rates and significantly reduced the dimensionality of the problems with tens of features. However, it is difficult to set an appropriate number of features especially in high-dimensional problems. Another multi-tree GP was also used in [29] to construct as many new features as original numeric features. DT was used in fitness function. Experiments on 10 UCI datasets with 5 to 60 features showed that the proposed method improve the performance of C4.5 on 8 datasets. However, on top of the high computational cost of wrapper approach, the GP representation of this method is not suitable for problems with thousands of features.

Another strategy for constructing multiple features is to use cooperative coevolution [19], where m concurrent populations of single-tree individuals were used to evolve m constructed features. Another method proposed in [13] also used single-tree GP to construct multiple features, however, without needing to predefine the number of constructed features. It included a special primitive function in the function set to automatically define a new feature based on the subtree under this function node. K-Nearest Neighbour was used to evaluate GP individuals.

Constructing one feature for each class of the problem is another approach. A multiple feature construction method using filter approach was proposed in [26] by running a single-tree GP program multiple times. Each time GP constructed one feature for each class. Constructed features were evaluated based on the impurity of the intervals using information measures which were formed by applying class dispersion to the transformed datasets.

Instead of using multiple GP runs, Ahmed et al proposed to use only one GP run to construct multiple features from all possible subtrees of the best individual [3]. Fisher criterion is used in fitness function to maximize the between-class scatter and minimize the within-class scatter of the constructed feature. P-value is also combined to ensure a significant separation between feature values. The method was evaluated on

mass spectrometry datasets with several hundreds to more than ten thousand features.

3 Methodology

To answer the raised questions, we use standard GP with an embedded approach to feature construction and selection on binary classification problem.

The GP algorithm includes a population of individuals. Each individual is a tree. Leaf nodes or terminal nodes of the tree are either original feature values or random constant values. Its internal nodes are operators or functions chosen from a predefined function set. Each individual is considered a constructed high-level feature because it can generate a new value from the original feature values.

An individual also works as a simple classifier that can classify binary problem: if an instance x has a negative value on the constructed high-level feature, GP will classify x to Class 1; otherwise to Class 2. The classification accuracies of GP will be used as a fitness measure to guide the search.

Because many of gene expression datasets have unbalanced data, the balanced accuracy [6,27] as shown in Equation 1 is used to evaluate the fitness of each GP individual. TP, TN, FN, FP are the numbers of true positive, true negative, false negative and false positive respectively. We choose the same weight 1/2 to treat the two classes equally important.

$$fitness = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \quad (1)$$

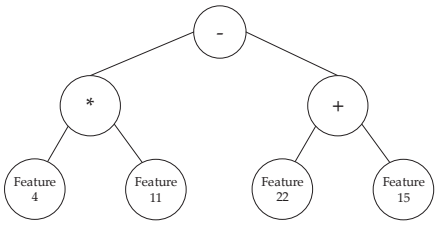

Fig. 1 An example of GP tree.

At the end of each GP run, the best GP tree is used to create six different feature sets. Figure 1 shows a simple GP tree, which uses Feature 4 (F_4), Feature 11 (F_{11}), Feature 22 (F_{22}), and Feature 4 (F_{15}) as the terminal nodes. We take Figure 1 as an example to illustrate how we generate six feature sets as follows:

1. Set 1: The single constructed feature only ("CF"), which is $F_0^{'} = F_4 * F_{11} - (F_{22} + F_{15})$;
2. Set 2: The original feature set augmented by the constructed feature ("FullCF"), which is $\{F_0^{'}, F_1, F_2,, ..., F_n\}$, where n is the total number of original features;
3. Set 3: Terminal feature set that are used to construct the new feature ("Ter"), which is $\{F_4, F_{11}, F_{22}, F_{15}\}$;
4. Set 4: The combination of Sets 1 and 3 ("CFTer"), which is $\{F_0^{'}, F_4, F_{11}, F_{22}, F_{15}\}$;
5. Set 5: Multiple constructed features from all possible subtrees of the GP tree ("mCF"), which is $\{F_0^{'}, F_1^{'}, F_2^{'}\}$, where $F_1^{'} = F_4 * F_{11}$, and $F_2^{'} = F_{22} + F_{15}$; and
6. Set 6: The combination of Sets 3 and 5 ("mCFTer"), which is $\{F_4, F_{11}, F_{22}, F_{15}, F_0^{'}, F_1^{'}, F_2^{'}\}$.

In general, the proposed method uses single-tree GP to generate six different feature subsets. It can be seen that Set 1 and Set 2 have been used in many existing methods [22,26]. Set 3 and Set 5 have been proposed in [3]. The other subsets have not been proposed. In addition, no investigation has been done to compare the performance of these sets. Further more, this method takes the embedded approach instead of filter as in many existing methods for feature construction and selection on high dimensional data. While filter methods are said to be faster than wrapper methods, their performance is usually not as good as wrappers. Embedded approach is a compromise of these two. Since a GP tree can be used as a classifier, GP can be employed as an embedded method for feature construction and selection.

Table 1 Description of seven microarray datasets

Dataset	# Features	#Instances	# Classes
Colon	2,000	62	2
DLBCL	5,469	77	2
Leukemia	7,129	72	2
CNS	7,129	60	2
Prostate	10,509	102	2
Breast	24,188	96	2
Ovarian	15,154	253	2

4 Experimental Design

Seven binary-class gene expression datasets[1] are used to test the performance of the six feature sets. The Breast dataset has a smaller number of features than the downloaded dataset since 293 features with identical values in all instances have been removed. One instance with more than 10,000 missing values is also deleted. Details

[1] http://www.gems-system.org, http://csse.szu.edu.cn/staff/zhuzx/Datasets.html

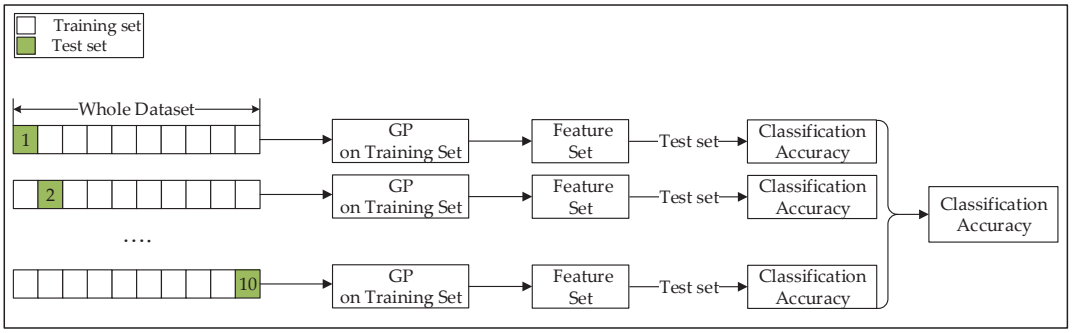

Fig. 2 GP for feature construction using 10-CV.

about the datasets used in this experiment are shown in Table 1.

All datasets are discretised in the same way as in [9] to reduce noise. Each feature is discretised into three category values (-1, 0 and 1) using mean (μ) and standard deviation (σ) of the feature values. A feature value x is set to 0 if it falls into the interval $[(\mu - \sigma)/2$ .. $(\mu + \sigma)/2]$. x will be set to -1 if $x < (\mu - \sigma)/2$ and set to 1 if $x > (\mu + \sigma)/2$.

Since the number of instances in these datasets are very small, we split the datasets into 10 folds and perform 10-fold cross validation (10-CV) to test the performance of GP for feature construction. Figure 2 shows the process of performing GP with 10-CV, where the same GP algorithm is run 10 times on different 10 training sets.

Since GP is a stochastic method, the process in Figure 2 is repeated for 30 independent runs with 30 different random seeds for GP. Therefore, totally 300 GP runs are executed. Note that performing experiments using this way is different from many papers in GP (or EC) for feature construction or selection [1,21,32], where GP is performed for 30 (or slightly more) times and 10-CV on the *whole dataset* is used in each fitness evaluation during the GP (or EC) evolutionary process. The results reported are "optimistically biased and are a subtle means of training on the test set" [15]. This issue is called *selection bias* in feature selection and has caught much attention from researchers [4,30]. Although the bias issue in feature construction has not been thoroughly studied, since feature selection and construction play a similar role in classification, the *construction bias* should also be seriously considered and avoided.

Table 2 describes the parameter settings in GP. The number of features in these datasets varies from thousands to tens of thousands; therefore, the search spaces of these problems are very different. As a result, the population size should be proportional to the number of features to explore more areas of the search space [5].

Table 2 Parameters in GP feature construction method

Parameters	**Parameter value**
Initial Population	Ramped Half-and Half
Maximum Tree Depth	17
Generations	50
Mutation Rate	0.2
Crossover Rate	0.8
Elitism	1
Population Size	#feature x β
Selection Method	Tournament Method
Tournament Size	7
Function set	$+$, $-$, $\times$, %, $\sqrt{}$, max, if
Terminal set	Features of a dataset, random constant value

However, due to the limitation of computer memory, we set the population size equal to the number of features multiplied by β where $\beta = \{3, 2, or 1\}$ if the number of features is less than 5,000, between 5,000 and 20,000, or more than 20,000 respectively.

The function set comprises of arithmetic operators ($+$, $-$, $\times$, %, $\sqrt{}$) in which protected division (%) results in zero when dividing by zero. $max(x_1, x_2)$ returns the maximum of the two inputs. if function takes three values and return the second if the first value is greater than zero, otherwise it returns the third value.

The performance of the feature sets that obtained from GP is tested on the *test set* using K-nearest Neighbour (KNN) (K=1), Naive Bayes (NB), Decision tree (DT) and GP as a classification algorithm (GPCA). GPCA is run 30 times for each created feature set on each dataset and the average classification accuracy is reported. Since the created feature sets are not as big as the original feature sets, the maximum tree depth is set to 5, the number of maximum generations is set to 30 and the population size is set to 2000. The Weka package [14] is used to run KNN (IB1), NB and DT (J48) with default settings. The ECJ library is used to run GP. Experiment runs on PC with Intel Core i7-4770 CPU @ 3.4GHz, running Ubuntu 4.6 and Java 1.7 with a total memory of 8GB.

In order to compare the classification accuracies of different GP created feature sets against using full feature set, in each dataset, a statistical significance test, Wilcoxon test, is performed with the significance level 0.05. Friedman test is also used to compare the performance of different feature sets for each learning algorithm over all datasets.

5 Results and Discussions

The results are shown in Tables 3 and 4, where "B" shows the best and "A±Std" shows the average and the standard deviation of the accuracy achieved through the 30 independent runs. "+" or "-" means the result is significantly better or worse than using all features and "=" means they are similar in Wilcoxon tests. The numbers under the dataset name is the number of instances in the dataset followed by the average CPU running time (in minutes) used by a single run comprising of 10 folds cross validation execution.

5.1 Created feature set size and GP running time

It can be seen from Table 3 that the average numbers of selected and constructed features are much smaller than the original feature number. These proportions are smaller than 1% on all datasets except Colon with 1% to 3%. Beside "CF" with the size of one, "Ter" is always the smallest among the created sets with the average size ranging from 10 in Ovarian to 35 in Breast dataset. The small size of the resulting feature sets is consistent with previous studies that although gene expression data has a large number of features, only a small number of features (less than 100 for two-class datasets) are relevant to the problems [33]. By using embedded approach, the running time of our method is relatively fast. This can be observed in the recorded running time in Table 3. The smallest dataset (Colon) takes about 6 minutes and the largest datasets (Ovarian) require about 5.25 hours to finish one run.

5.2 Training Results

To analyse the performance of GP and different generated feature sets, first we show the training results in Table 3. These accuracies are obtained by firstly using each of the six features sets to transform the training set (i.e. 9 folds), then applying each classification algorithm on these transformed training set. The best and average training accuracies of the learnt models over 30 runs are reported.

KNN is a lazy learning algorithm, in theory, its training accuracies should be 100%. However, it can be seen that the accuracies of using the constructed feature on five datasets are less than 100%. The reason of this strange phenomenon is that the constructed features have the same values for multiple instances which belong to different classes. These inconsistent instances obviously affect the quality of these feature sets as well as the learning algorithm performance. Solution of this problem should be further investigated. One possible solution can be applying penalty to these GP individuals during the evolutionary process.

For NB algorithm, all the six created sets can either obtain similar or significant better accuracies than using all features on six out of seven datasets. Among these feature sets, "Full" and "FullCF" have the worst results on all datasets. This indicates that these datasets contain many redundant or correlated features which degrade NB performance because the conditional independence assumption no longer holds [28]. On the Prostate dataset, NB can achieve an increase of 25% in accuracy using GP selected features and 30% using only the constructed feature. The results show that GP has the ability to select only informative features and construct better discriminating features. Using these features, NB can significantly improve its performance on datasets which have many correlated features.

Similarly, the created feature sets also help DT and GP achieve nearly maximum classification accuracies. However, unlike in NB, not all the created sets can outperform the results of DT and GP using full feature set. A clear pattern can be seen in Table 3 with the minus signs shown in parentheses consistently appear in the "Ter" rows of DT column, as well as in the "FullCF" and "Ter" rows of GP column. It is also observed in NB that "Ter" accuracies on all datasets are always smaller than the other four created sets which comprise the constructed feature ("CF", "CFTer", "mCF" and "mCFTer"). This indicates that the constructed feature perform better than the selected features on the training data.

In general, the training results from NB, DT and GP show that GP has the ability to select informative features and construct new high-level features that provide a much better discriminating power than the original features.

5.3 Test Results

The test accuracy is achieved by firstly transforming the training and test sets according to the six feature sets. Then a classifier is learnt based on the transformed training set and tested on the transformed test set. The

Table 3 Training Results

Dataset	Subset	#F	A±Std-KNN	B-NB	A±Std-NB	B-DT	A±Std-DT	B-GP	A±Std-GP
	Full	2000	100.0 ±0.00	84.96	84.96 ±0.00	97.13	97.13 ±0.00	100.0	99.87 ±0.21
	CF	1	99.83 ±0.47 –	97.14	91.62 ±3.87 +	100.0	99.87 ±0.21 +	100.0	99.88 ±0.21 =
Colon	FullCF	2001	100.0 ±0.00 =	86.75	85.65 ±0.37 +	100.0	99.92 ±0.14 +	99.88	99.74 ±0.11 (–)
(62)	Ter	22	100.0 ±0.00 =	89.80	87.17 ±1.58 +	96.24	94.21 ±1.07 (–)	96.62	95.65 ±0.42 (–)
5.89(m)	CFTer	23	100.0 ±0.00 =	94.45	92.12 ±1.57 +	100.0	99.87 ±0.21 +	100.0	99.91 ±0.18 +
	mCF	37	99.99 ±0.05 =	94.28	92.58 ±1.42 +	100.0	99.87 ±0.21 +	100.0	99.91 ±0.17 +
	mCFTer	59	100.0 ±0.00 =	94.10	92.57 ±1.13 +	100.0	99.87 ±0.21 +	100.0	99.91 ±0.17 +
	Full	5469	100.0 ±0.00	90.91	90.91 ±0.00	98.85	98.85 ±0.00	100.0	100.0 ±0.00
	CF	1	100.0 ±0.00 =	100.0	98.24 ±1.15 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
DLBCL	FullCF	5470	100.0 ±0.00 =	91.34	91.02 ±0.11 +	100.0	100.0 ±0.00 +	99.99	99.97 ±0.01 (–)
(77)	Ter	15	100.0 ±0.00 =	98.70	96.43 ±1.23 +	98.70	97.13 ±0.57 (–)	99.59	98.91 ±0.42 (–)
12.35(m)	CFTer	16	100.0 ±0.00 =	99.71	98.36 ±0.81 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCF	25	100.0 ±0.00 =	98.85	97.35 ±1.00 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCFTer	40	100.0 ±0.00 =	98.99	97.58 ±1.00 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	Full	7129	100.0 ±0.00	98.15	98.15 ±0.00	99.38	99.38 ±0.00	100.0	100.0 ±0.00
	CF	1	100.0 ±0.00 =	99.54	96.93 ±3.30 =	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
Leukemia	FullCF	7130	100.0 ±0.00 =	98.30	98.28 ±0.05 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
(72)	Ter	12	100.0 ±0.00 =	98.61	97.07 ±0.92 –	98.92	97.63 ±0.66 (–)	99.85	99.35 ±0.31 (–)
14.22(m)	CFTer	13	100.0 ±0.00 =	99.54	98.28 ±0.66 =	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCF	19	100.0 ±0.00 =	98.77	97.54 ±0.92 –	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCFTer	31	100.0 ±0.00 =	99.08	98.03 ±0.68 =	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	Full	7129	100.0 ±0.00	73.89	73.89 ±0.00	98.71	98.70 ±0.00	100.0	99.96 ±0.09
	CF	1	99.96 ±0.09 –	97.04	90.14 ±3.44 +	100.0	99.96 ±0.09 +	100.0	99.96 ±0.09 =
CNS	FullCF	7130	100.0 ±0.00 =	75.18	74.26 ±0.30 +	100.0	99.97 ±0.07 +	99.75	99.65 ±0.05 (–)
(60)	Ter	30	100.0 ±0.00 =	87.78	81.68 ±2.06 +	96.30	94.20 ±1.08 (–)	93.50	92.53 ±0.49 (–)
19.42(m)	CFTer	31	100.0 ±0.00 =	92.78	90.02 ±1.84 +	100.0	99.97 ±0.07 +	100.0	99.98 ±0.04 +
	mCF	48	100.0 ±0.00 =	93.70	90.99 ±1.85 +	100.0	99.97 ±0.07 +	100.0	99.98 ±0.05 +
	mCFTer	78	100.0 ±0.00 =	93.70	90.35 ±1.81 +	100.0	99.97 ±0.07 +	100.0	99.98 ±0.06 +
	Full	10509	100.0 ±0.00	66.67	66.67 ±0.00	98.59	98.59 ±0.00	100.0	99.84 ±0.18
	CF	1	99.75 ±0.77 –	99.46	96.52 ±2.45 +	100.0	99.84 ±0.18 +	100.0	99.84 ±0.18 =
Prostate	FullCF	10510	100.0 ±0.00 =	67.32	66.97 ±0.11 +	100.0	99.84 ±0.18 +	99.83	99.71 ±0.11 (–)
(102)	Ter	22	99.99 ±0.04 =	93.58	91.42 ±1.02 +	97.17	96.17 ±0.62 (–)	98.54	97.54 ±0.51 (–)
71.37(m)	CFTer	23	99.99 ±0.04 =	96.73	94.31 ±0.97 +	100.0	99.84 ±0.18 +	100.0	99.85 ±0.17 +
	mCF	35	99.98 ±0.10 =	97.93	95.11 ±0.91 +	100.0	99.84 ±0.18 +	100.0	99.85 ±0.17 +
	mCFTer	57	99.99 ±0.04 =	97.39	94.85 ±0.85 +	100.0	99.84 ±0.18 +	100.0	99.85 ±0.17 +
	Full	24188	100.0 ±0.00	88.31	88.31 ±0.00	98.03	98.03 ±0.00	100.0	99.97 ±0.06
	CF	1	99.98 ±0.07 =	98.26	92.68 ±3.69 +	100.0	99.97 ±0.06 +	100.0	99.97 ±0.06 =
Breast	FullCF	24189	100.0 ±0.00 =	88.42	88.32 ±0.04 =	100.0	99.98 ±0.05 +	99.35	99.17 ±0.09 (–)
(96)	Ter	34	100.0 ±0.00 =	88.78	86.20 ±1.12 –	95.95	94.87 ±0.76 (–)	93.80	91.60 ±0.81 (–)
294.41(m)	CFTer	35	100.0 ±0.00 =	94.10	92.01 ±1.54 +	100.0	99.97 ±0.06 +	100.0	99.98 ±0.05 +
	mCF	60	100.0 ±0.00 =	95.14	91.31 ±1.98 +	100.0	99.97 ±0.06 +	100.0	99.98 ±0.05 +
	mCFTer	95	100.0 ±0.00 =	94.80	91.52 ±1.63 +	100.0	99.97 ±0.06 +	100.0	99.98 ±0.05 =
	Full	15154	100.0 ±0.00	91.79	91.79 ±0.00	99.91	99.91 ±0.00	100.0	100.0 ±0.00
	CF	1	100.0 ±0.00 =	99.91	99.22 ±0.95 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
Ovarian	FullCF	15155	100.0 ±0.00 =	91.92	91.82 ±0.03 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 (–)
(253)	Ter	9	100.0 ±0.00 =	99.17	98.63 ±0.29 +	99.82	99.29 ±0.29 (–)	100.0	99.91 ±0.09 (–)
325.77(m)	CFTer	10	100.0 ±0.00 =	99.82	99.28 ±0.27 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCF	16	100.0 ±0.00 =	99.87	98.89 ±0.49 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =
	mCFTer	25	100.0 ±0.00 =	99.65	99.05 ±0.34 +	100.0	100.0 ±0.00 +	100.0	100.0 ±0.00 =

best and average accuracies of 30 runs are reported in Table 4.

For KNN, the selected and/or constructed features can either maintain or improve the performance of KNN with a much smaller number of features. Using only one constructed feature, KNN can achieve significant better accuracies than using all features on four datasets, similar accuracies on the two datasets. Only in Colon dataset, it has lower accuracy than using all features but its best accuracy is 5% higher than using all features. We also see that the "FullCF" always have the same results as using all features. This is trivial because adding one more feature to thousands of features cannot make any difference for KNN. Among the six created sets, "CFTer", which is the constructed feature combined with terminal features, obtains the best accuracies on six out of seven datasets. Similarly, this feature set also helps NB achieve the highest results on five datasets with a maximum increment of 27% in Prostate dataset. In general, the created sets can improve the performance of NB on most datasets except Leukemia and Breast.

For DT and GP, all the six created sets obtain similar accuracies on each dataset. While GP either has similar or slightly better accuracies than using all features on all datasets, DT increases 6% and 7% on DLBCL and CNS datasets, and decreases 1 to 3% on the remaining five datasets. However, the best accuracies obtained by DT using all the six sets are always higher than using all features with a maximum difference of 23% in CNS dataset. It is noticed that when using feature sets containing the constructed feature, DT always use this feature in its learnt models. This indicates that the constructed feature is the best splitting feature according to DT feature selection criterion. This explains

Table 4 Test Results

Dataset	Subset	#F	B-KNN	A±Std-KNN	B-NB	A±Std-NB	B-DT	A±Std-DT	B-GP	A±Std-GP
Colon (62)	Full	2000	74.28	74.28 ±0.00	72.62	72.62 ±0.00	74.29	**74.29 ±0.00**	79.28	71.82 ±4.55
	CF	1	79.28	71.40 ±4.46 –	78.81	69.64 ±4.17 –	79.28	72.25 ±4.07 –	79.34	71.80 ±4.50 =
	FullCF	2001	74.28	74.28 ±0.00 =	74.29	72.59 ±0.92 =	79.28	72.25 ±4.07 –	79.12	72.02 ±4.06 =
	Ter	22	85.47	76.40 ±4.82 +	85.48	75.81 ±3.95 +	80.95	73.32 ±4.18 =	79.02	**74.37 ±2.40 +**
	CFTer	23	87.38	**76.90 ±5.21 +**	87.14	**75.96 ±4.03 +**	79.28	72.25 ±4.07 –	79.12	71.76 ±4.42 =
	mCF	37	80.95	71.24 ±4.73 –	87.14	73.56 ±4.94 =	79.28	72.25 ±4.07 –	79.45	71.75 ±4.48 =
	mCFTer	59	84.05	75.05 ±3.85 =	87.14	74.66 ±4.31 +	79.28	72.25 ±4.07 –	79.20	71.79 ±4.40 =
DLBCL (77)	Full	5469	84.46	84.46 ±0.00	81.96	81.96 ±0.00	80.89	80.89 ±0.00	94.64	86.34 ±4.17
	CF	1	96.07	86.65 ±3.76 +	92.32	86.27 ±4.28 +	94.64	86.51 ±4.08 +	94.72	86.37 ±4.08 =
	FullCF	5470	84.46	84.46 ±0.00 =	81.96	81.96 ±0.00 =	94.64	86.51 ±4.08 +	92.88	86.74 ±2.82 =
	Ter	15	95.00	86.36 ±4.13 +	96.25	88.49 ±3.49 +	98.75	85.04 ±5.45 +	94.26	**87.28 ±3.80 =**
	CFTer	16	95.00	**86.80 ±4.83 +**	96.07	**89.36 ±4.00 +**	94.64	**86.51 ±4.08 +**	94.77	86.33 ±4.07 =
	mCF	25	92.50	85.03 ±4.39 =	93.39	87.50 ±4.07 +	93.39	86.30 ±4.06 +	93.98	86.29 ±3.98 =
	mCFTer	40	92.32	86.14 ±3.40 +	93.75	88.43 ±3.71 +	93.39	86.30 ±4.06 +	94.01	86.27 ±3.99 =
Leukemia (72)	Full	7129	88.57	88.57 ±0.00	91.96	91.96 ±0.00	91.61	**91.61 ±0.00**	94.46	88.89 ±2.83
	CF	1	94.46	89.03 ±2.71 =	93.21	87.26 ±4.44 –	95.89	88.97 ±2.96 –	94.56	88.84 ±2.78 =
	FullCF	7130	88.57	88.57 ±0.00 =	93.21	92.01 ±0.23 =	95.89	88.97 ±2.96 –	93.71	89.11 ±2.23 =
	Ter	12	95.89	89.39 ±3.53 =	96.07	**92.24 ±2.69 =**	98.75	89.85 ±4.32 =	96.43	**89.98 ±2.81 +**
	CFTer	13	97.32	**90.28 ±3.58 +**	97.32	91.46 ±2.91 =	95.89	88.97 ±2.96 –	94.43	88.76 ±2.74 –
	mCF	19	93.39	86.71 ±4.48 –	96.07	88.89 ±3.79 –	94.46	89.01 ±2.69 –	94.52	88.92 ±2.67 =
	mCFTer	31	95.89	89.08 ±3.99 =	96.07	90.34 ±3.43 –	94.46	89.01 ±2.69 –	94.61	88.86 ±2.71 =
CNS (60)	Full	7129	56.67	56.67 ±0.00	58.33	58.33 ±0.00	50.00	50.00 ±0.00	70.00	57.44 ±6.31
	CF	1	70.00	57.56 ±5.87 =	70.00	58.44 ±5.94 =	70.00	**57.78 ±6.05 +**	69.94	57.46 ±6.23 =
	FullCF	7130	56.67	56.67 ±0.00 =	60.00	58.44 ±0.42 =	70.00	**57.78 ±6.05 +**	67.61	57.12 ±5.44 =
	Ter	30	70.00	57.56 ±6.09 =	70.00	59.89 ±3.86 +	73.33	**57.78 ±5.63 +**	63.56	56.42 ±3.07 =
	CFTer	31	73.33	57.33 ±6.25 =	70.00	**60.22 ±4.85 +**	70.00	**57.78 ±6.05 +**	70.78	**57.53 ±6.26 =**
	mCF	48	71.67	57.44 ±6.73 =	70.00	58.94 ±7.10 =	70.00	**57.78 ±6.05 +**	70.44	57.51 ±6.26 =
	mCFTer	78	70.00	**58.56 ±6.83 =**	71.67	58.78 ±6.31 =	70.00	**57.78 ±6.05 +**	70.28	57.52 ±6.20 =
Prostate (102)	Full	10509	81.55	81.55 ±0.00	60.55	60.55 ±0.00	86.18	**86.18 ±0.00**	91.18	83.96 ±3.08
	CF	1	90.18	83.72 ±3.18 +	90.18	83.18 ±3.68 +	90.18	83.82 ±2.85 –	91.08	83.96 ±3.04 =
	FullCF	10510	81.55	81.55 ±0.00 =	60.55	60.55 ±0.00 =	90.18	83.82 ±2.85 –	90.24	84.20 ±2.55 +
	Ter	22	90.36	83.05 ±3.77 +	90.27	87.04 ±2.06 +	90.18	82.32 ±3.39 –	90.84	**85.69 ±1.87 +**
	CFTer	23	90.36	**84.09 ±3.71 +**	90.36	**87.07 ±2.52 +**	90.18	83.82 ±2.85 –	91.02	83.93 ±3.01 =
	mCF	35	88.27	82.84 ±2.39 +	90.27	85.37 ±2.45 +	90.18	83.82 ±2.85 –	90.88	83.91 ±2.98 =
	mCFTer	57	89.27	83.50 ±2.89 +	91.18	86.43 ±2.65 +	90.18	83.82 ±2.85 –	90.98	83.91 ±3.00 =
Breast (96)	Full	24188	57.78	57.78 ±0.00	74.89	74.89 ±0.00	63.56	**63.56 ±0.00**	71.56	60.55 ±5.49
	CF	1	70.78	60.59 ±5.37 +	71.78	60.26 ±5.63 –	70.78	60.49 ±5.33 –	71.56	60.54 ±5.51 =
	FullCF	24189	57.78	57.78 ±0.00 =	75.89	**75.32 ±0.51 +**	70.78	60.53 ±5.29 –	69.15	60.84 ±4.44 =
	Ter	34	72.11	61.40 ±4.73 +	73.67	67.73 ±2.91 –	70.67	61.19 ±4.93 –	67.94	**63.20 ±2.42 +**
	CFTer	35	71.00	**61.68 ±4.52 +**	73.67	66.30 ±4.57 –	70.78	60.49 ±5.33 –	71.46	60.58 ±5.47 =
	mCF	60	70.56	60.96 ±4.99 +	73.67	63.33 ±4.32 –	70.78	60.53 ±5.32 –	71.46	60.57 ±5.45 =
	mCFTer	95	68.44	61.16 ±3.81 +	72.67	64.44 ±4.39 –	70.78	60.53 ±5.32 –	71.66	60.54 ±5.48 =
Ovarian (253)	Full	15154	91.28	91.28 ±0.00	90.05	90.05 ±0.00	98.41	**98.41 ±0.00**	99.62	97.86 ±1.21
	CF	1	99.62	97.86 ±1.22 +	99.62	97.22 ±1.48 +	99.62	97.89 ±1.18 –	99.60	97.86 ±1.21 =
	FullCF	15155	91.28	91.28 ±0.00 =	90.05	90.05 ±0.00 =	99.62	97.89 ±1.18 –	99.25	**97.98 ±0.93 =**
	Ter	9	100.0	98.15 ±0.96 +	98.82	97.75 ±0.68 +	100.0	97.87 ±1.08 –	99.60	97.92 ±0.88 =
	CFTer	10	100.0	**98.42 ±0.99 +**	99.62	**98.20 ±0.89 +**	99.62	97.89 ±1.18 –	99.56	97.85 ±1.19 =
	mCF	16	99.60	97.41 ±1.24 +	99.62	97.40 ±1.01 +	99.62	97.84 ±1.19 –	99.59	97.84 ±1.20 =
	mCFTer	25	99.62	98.09 ±1.03 +	99.60	97.65 ±0.94 +	99.62	97.84 ±1.19 –	99.52	97.85 ±1.18 =

why DT accuracies obtained by these created sets are very similar to GP "CF" accuracies.

5.3.1 Comparison between the best and the average accuracy

In most datasets, the differences between the best and the average results are quite high. For example, the best result NB achieve using "CFTer" is 75.96% on the Colon dataset which is 12% lower than its best result (87.14%). This gap is even higher than 15% in the DT results on the CNS dataset. The results indicate that the learning algorithms may perform well on some test folds and poorly on some other folds. To see if this is the case, we look at the accuracies of each classifier on each data fold. To leave out the effect of feature selection or construction, we use the result of using full feature set. Table 5 show the "Full" accuracies of Colon dataset with KNN, NB and DT which are deterministic learning algorithms. It can be seen that the gap between the obtain accuracies in different folds is very high with the maximum of 50% in NB.

Table 5 Results of "Full" feature set on each fold of Colon dataset

Fold	KNN	NB	DT
0	57.14	71.43	71.43
1	85.71	71.43	71.43
2	83.33	50.00	83.33
3	83.33	100.0	83.33
4	66.67	66.67	66.67
5	83.33	50.00	83.33
6	83.33	83.33	66.67
7	83.33	100.0	66.67
8	50.00	66.67	66.67
9	66.67	66.67	83.33
Max difference	**35.71**	**50.00**	**16.66**

This result shows that some test folds may have a very different distribution to their corresponding training folds. As a result, it is difficult for learning algorithms to learn a model that can perform well on the test folds. We will further investigate this problem in Section 5.6. In addition, with a small number of instances in one test fold (less than ten instances in most datasets), one misclassified instance can significantly decrease the classification accuracies. This explains why datasets with the smallest number of instances, such as CNS with 60 and Colon 62 instances, have the biggest difference between the best and the average accuracies. In contrast, this gap in Ovarian dataset with 253 instances is only about 2%.

It is also noticed that while the selected feature set ("Ter") performs worse than other sets on the training data, it achieves the highest average accuracies on five out of seven datasets in GP and performs better than other sets on Colon and Leukemia in DT. This result is in contrast with the observation in the training results. This phenomenon suggests that the constructed features are overfitting to the training data. This overfitting problem is clearer when observing the big differences between the training and test accuracies. However, these gaps vary between different datasets. In "CF" set, this difference can be as small as 3% in Ovarian, or as large as 27% in Colon, or even very large as in CNS with 42%. This indicates that this overfitting problem is related to the characteristics of the dataset. We will further analyse this in Section 5.6.

5.4 Comparison between different created feature sets

In Table 4, the highest average accuracy among different created feature sets of the same classifier on a certain dataset (the best in a cell) is shown in bold. It can be seen that the "CFTer" which combines the constructed feature with terminal features seems to outperform other feature sets in both KNN and NB on most datasets. However, to confirm if it is really significantly better than others, we perform Friedman test with Tukey as the post hoc test using R package. The results show that there is no significant difference between different created sets in DT and GP. This again confirms that the constructed feature achieves similar performance as other sets with a much higher number of features in these two learning algorithms. On the other hand, a significant difference between these feature sets is found in KNN with $p\text{-}value = 0.01$ and NB with $p\text{-}value < 0.01$. Figure 3 and Figure 4 show the boxplots of the differences between pairs of created feature sets in KNN and NB results respectively. In these figures, the "Full" and the six created sets are indexed from zero to six with the same order as shown in Table 4. If the difference between two feature sets is significant with $p\text{-}value < 0.5$, its corresponding boxplot is filled.

The first five columns in these two figures show that in average all the six created sets have similar or better results than the full feature set even though not all of these differences are significant. For KNN, "CFTer" obtains significant better result than "Full", "FullCF" and "mCF" with $p\text{-}value < 0.05$. This means that among the six created sets, combination of the constructed and selected features help KNN achieve its best performance. For NB, "CFTer" and "Ter" sets outperform the constructed feature. These results again confirm the ability of GP in selecting informative features and constructing new features that can significantly reduce the feature set size while improve or at least maintain the classification performance of these learning algorithms.

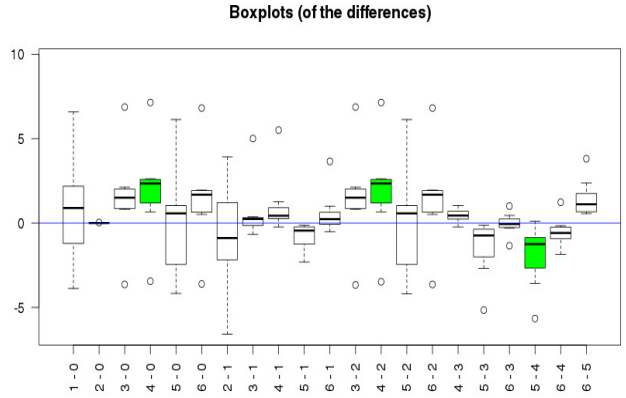

Fig. 3 Freidman Post-hoc test for KNN on seven datasets.

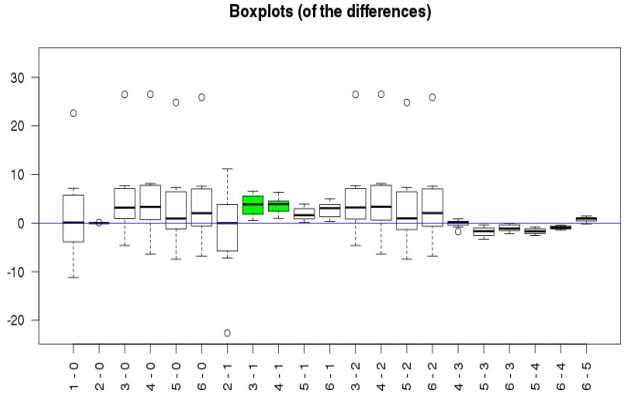

Fig. 4 Freidman Post-hoc test for NB on seven datasets.

5.5 Constructed feature

To see why the constructed and selected features can achieve good performance, we take a constructed feature in a GP run on the DLBCL dataset as a typical example to make analysis. Figure 5 shows the GP tree of a constructed feature in DLBCL dataset. It is constructed from three original features which are feature F1156, F1259, and F3228. The values of these three features and the constructed feature are plotted in Figure 6 and Figure 7. It can be seen from these scatter plots that these selected features have low impurity with one nominal value already belong to one class. By combining the three original features, the constructed feature splits instances in the two classes into two completely separate intervals. Therefore, using this constructed feature, the GP classifier can easily classify an instance x by executing the following rule:
$IF\ \ constructedF <= 0\ \ THEN\ \ x\ \in\ class_0,$
$ELSE\ \ x\ \in\ class_1$

The results show that GP has the ability to select informative features to build high-level features with a higher discriminating ability.

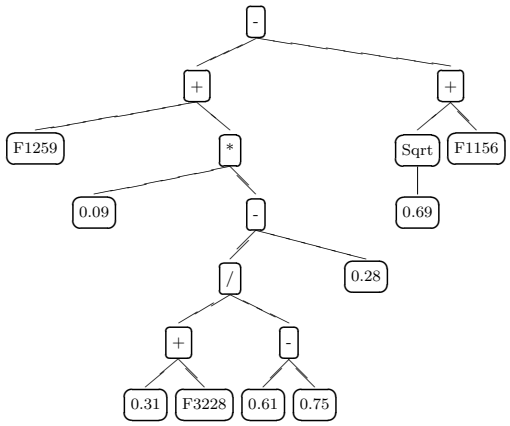

Fig. 5 DLBCL constructed feature.

5.6 Overfitting problem

The overfitting problem that we have seen in the results has different effects on different datasets. Therefore, to analyse this problem, we look at the distributions of each feature in these datasets. All features in the six out of seven datasets have a skew distribution with many outliers. We take Colon as an example. Figure 8 shows the boxplot of its first 50 features. We can see that these features have a skewed distribution. Each feature has many outliers scattering far away from its mean value. In the experiments, Colon is divided into

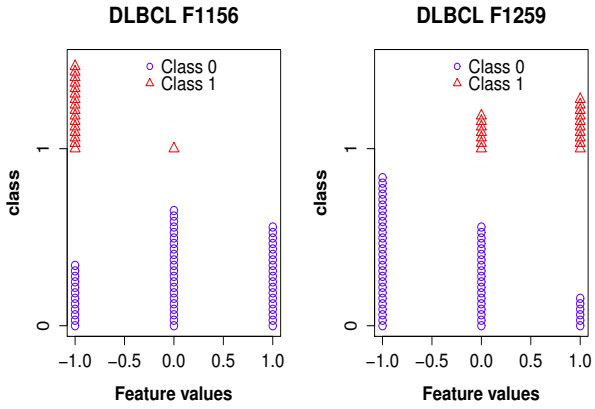

Fig. 6 DLBCL Feature F1156 and F1259.

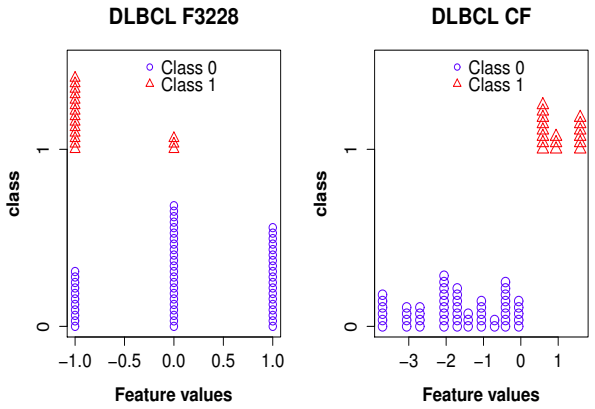

Fig. 7 DLBCL Feature F3228 and the Constructed Feature.

10 folds each of which has about 6 instances. Therefore, it is likely that the distributions of the training and the test folds are very different. As a result, the constructed or selected features based on the training fold cannot generalise well to correctly predict the unseen data in the test fold. This may be the reason why the training and test accuracies are so different. This explanation is concordant to the result of Ovarian dataset where all of the learning algorithms achieve similar performance on training and test sets. The boxplot of the first 50 features of Ovarian in Figure 9 shows that these features have a rather symmetric distribution without many outliers. Besides outliers, the small number of instances is another reason for overfitting problem. It is difficult for learning algorithms to generalise a good model from a small set of examples. This explains why the overfitting problem has more effect in Colon and CNS than other datasets. It also suggests that using mean and standard deviation for such data with many outliers might not be an ideal method for discretisation and we will make further investigation in the future.

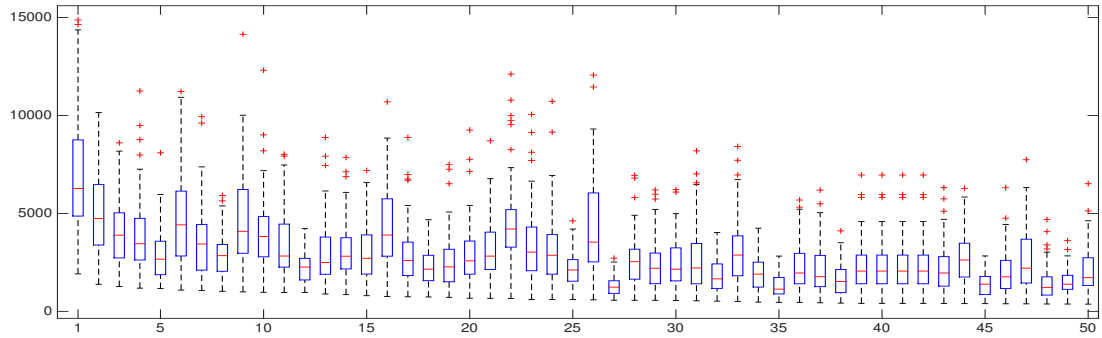

Fig. 8 Colon: first 50 features.

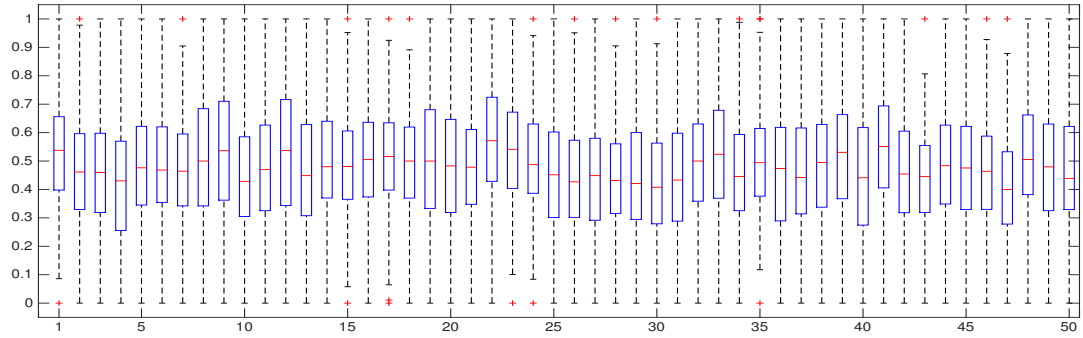

Fig. 9 Ovarian: first 50 features.

6 Conclusions and Future Work

This paper investigates the use of GP for feature construction and selection on high-dimensional data by analysing the performance of six different sets of constructed and/or selected features on four different classification algorithms. The experiments on seven binary-class datasets show that in most cases, the features constructed or selected by GP can improve the performance of KNN and NB with a much smaller feature set. The constructed features in general can work as good as other GP created feature sets to maintain the performance of DT and GP classifiers. Further analysis on the constructed feature shows that by choosing informative features, GP can construct new features which have better discriminating ability than original features. The difference between the training and test results on some datasets indicates the problem of overfitting. By analysing the datasets, it is found that this problem occurs when the data has a skewed distribution with many outliers. The fewer instances the dataset, the worse the overfitting problem.

Although the constructed feature combined with terminal features seems to be the best feature set in improving learning performance, this feature set size may be still too small to effectively classify the problem. Increasing the size of this feature set may further improve the classification performance on high-dimensional data. Results have shown GP's potential in selecting and constructing features with better discriminating ability. Comparison between GP and other algorithms addressing the similar problems should be made to better promote GP use in feature selection and construction. In this study, only binary problems have been used to test the performance of GP. GP effectiveness should also be tested on multi-class problems. Our future work will focus on these directions and we will also work on solving the overfitting problem.

References

1. Ahmed, S., Zhang, M., Peng, L.: Genetic programming for biomarker detection in mass spectrometry data. In: Advances in Artificial Intelligence, *Lecture Notes in Computer Science*, vol. 7691, pp. 266–278 (2012)
2. Ahmed, S., Zhang, M., Peng, L.: Enhanced feature selection for biomarker discovery in lc-ms data using gp. In: IEEE Congress on Evolutionary Computation (CEC'13), pp. 584–591 (2013)
3. Ahmed, S., Zhang, M., Peng, L., Xue, B.: Multiple feature construction for effective biomarker identification

and classification using genetic programming. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, GECCO '14, pp. 249–256. ACM (2014)
4. Ambroise, C., McLachlan, G.J.: Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the National Academy of Sciences **99**(10), 6562–6566 (2002)
5. Banzhaf, W., Francone, F.D., Keller, R.E., Nordin, P.: Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers Inc. (1998)
6. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Reusing genetic programming for ensemble selection in classification of unbalanced data. Evolutionary Computation, IEEE Transactions on **18**(6), 893–908 (2014)
7. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers & Electrical Engineering **40**, 16–28 (2014)
8. De Stefano, C., Fontanella, F., Marrocco, C., di Freca, A.S.: A GA-based feature selection approach with an application to handwritten character recognition. Pattern Recognition Letters **35**, 130–141 (2014)
9. Ding, C., Peng, H.: Minimum redundancy feature selection from microarray gene expression data. Journal of Bioinformatics and Computational Biology **03**(02), 185–205 (2005)
10. Espejo, P., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **40**(2), 121–144 (2010). DOI 10.1109/TSMCC.2009.2033566
11. Estébanez, C., Valls, J.M., Aler, R.: Gppe: a method to generate ad-hoc feature extractors for prediction in financial domains. Applied Intelligence **29**(2), 174–185 (2008)
12. Guo, H., Nandi, A.K.: Breast cancer diagnosis using genetic programming generated feature. Pattern Recognition **39**(5), 980–987 (2006)
13. Guo, L., Rivero, D., Dorado, J., Munteanu, C.R., Pazos, A.: Automatic feature extraction using genetic programming: An application to epileptic eeg classification. Expert Systems with Applications **38**(8), 10,425–10,436 (2011)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations **11**, 931–934 (2009)
15. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence **97**, 273–324 (1997)
16. Krawiec, K.: Genetic programming-based construction of features for machine learning and knowledge discovery tasks. Genetic Programming and Evolvable Machines **3**, 329–343 (2002)
17. Krawiec, K.: Evolutionary feature selection and construction. In: Encyclopedia of Machine Learning, pp. 353–357. Springer (2010)
18. Langdon, W.B., Buxton, B.F.: Genetic programming for mining dna chip data from cancer patients. Genetic Programming and Evolvable Machines **5**(3), 251–257 (2004)
19. Lin, Y., Bhanu, B.: Evolutionary feature synthesis for object recognition. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **35**(2), 156–171 (2005)
20. Lones, M., Smith, S.L., Alty, J.E., Lacy, S.E., Possin, K.L., Jamieson, D., Tyrrell, A.M., et al.: Evolving classifiers to recognize the movement characteristics of parkinson's disease patients. Evolutionary Computation, IEEE Transactions on **18**(4), 559–576 (2014)
21. Mohamad, M., Omatu, S., Deris, S., Yoshioka, M., Abdullah, A., Ibrahim, Z.: An enhancement of binary particle swarm optimization for gene selection in classifying cancer classes. Algorithms for Molecular Biology **8**(1), 15 (2013)
22. Muharram, M., Smith, G.: Evolutionary constructive induction. IEEE Transactions on Knowledge and Data Engineering **17**, 1518–1528 (2005)
23. Nekkaa, M., Boughaci, D.: A memetic algorithm with support vector machine for feature selection and classification. Memetic Computing **7**(1), 59–73 (2015)
24. Neshatian, K., Zhang, M.: Dimensionality reduction in face detection: A genetic programming approach. In: 24th International Conference on Image and Vision Computing, pp. 391–396 (2009)
25. Neshatian, K., Zhang, M.: Using genetic programming for context-sensitive feature scoring in classification problems. Connection Science **23**(3), 183–207 (2011)
26. Neshatian, K., Zhang, M., Andreae, P.: A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. IEEE Transactions on Evolutionary Computation **16**(5), 645–661 (2012)
27. Patterson, G., Zhang, M.: Fitness functions in genetic programming for classification with unbalanced data. In: AI 2007: Advances in Artificial Intelligence, pp. 769–775. Springer (2007)
28. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press (2009)
29. Smith, M., Bull, L.: Genetic Programming with a Genetic Algorithm for Feature Construction and Selection. Genetic Programming and Evolvable Machines **6**, 265–281 (2005)
30. Statnikov, A., Aliferis, C.F., Tsamardinos, I., Hardin, D., Levy, S.: A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. Bioinformatics **21**, 631–643 (2005)
31. Wang, P., Emmerich, M., Li, R., Tang, K., Back, T., Yao, X.: Convex hull-based multiobjective genetic programming for maximizing receiver operating characteristic performance. Evolutionary Computation, IEEE Transactions on **19**(2), 188–200 (2015)
32. Yu, H., Gu, G., Liu, H., Shen, J., Zhao, J.: A modified ant colony optimization algorithm for tumor marker gene selection. Genomics, Proteomics & Bioinformatics **7**(4), 200–208 (2009)
33. Zhu, Z., Ong, Y.S., Dash, M.: Markov blanket-embedded genetic algorithm for gene selection. Pattern Recognition **40**(11), 3236–3248 (2007)