# Automatically Evolving Difficult Benchmark Feature Selection Datasets with Genetic Programming

Andrew Lensen
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
andrew.lensen@ecs.vuw.ac.nz

Bing Xue
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
bing.xue@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

## ABSTRACT

There has been a wealth of feature selection algorithms proposed in recent years, each of which claims superior performance in turn. A wide range of datasets have been used to compare these algorithms, each with different characteristics and quantities of redundant and noisy features. Hence, it is very difficult to comprehensively and fairly compare these feature selection methods in order to find which are most robust and effective. In this work, we examine using Genetic Programming to automatically synthesise redundant features for augmenting existing datasets in order to more scientifically test feature selection performance. We develop a method for producing complex multi-variate redundancies, and present a novel and intuitive approach to ensuring a range of redundancy relationships are automatically created. The application of these augmented datasets to well-established feature selection algorithms shows a number of interesting and useful results and suggests promising directions for future research in this area.

## CCS CONCEPTS

• **Computing methodologies** → **Feature selection**; *Unsupervised learning*; • **Software and its engineering** → **Genetic programming** ;

## KEYWORDS

Genetic Programming, Feature Creation, Feature Selection, Mutual Information

**ACM Reference Format:**
Andrew Lensen, Bing Xue, and Mengjie Zhang. 2018. Automatically Evolving Difficult Benchmark Feature Selection Datasets with Genetic Programming. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205552

## 1 INTRODUCTION

As the modern world rapidly becomes more data-centric, datasets continue to grow in both width and length. However, many common machine learning algorithms are unable to scale effectively as the number of features grows. Feature Selection (FS) algorithms are being increasingly used as a method for reducing the dimensionality of datasets in order to improve the quality and interpretability of machine learning results [11]. Such FS algorithms primarily work by identifying and removing *irrelevant* or *redundant* features from a feature set [17]. Irrelevant (or *noisy*) features provide little useful information about the instances of a dataset, or may even "mislead" the machine learning algorithm. While removing irrelevant features is reasonably straightforward, redundant features (*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 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!) [11]. 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, it is not obvious how to add features with complex redundancies.

Previous work has added r.fs to an existing dataset by scaling and offsetting features in the feature space, i.e. creating "new" features of the form $F_{new} = \alpha \times F_{existing} + \beta$, for some real $\alpha, \beta$. Such features are univariately and linearly redundant and do not present much of a challenge for a FS algorithm. Recently, we proposed a Genetic Programming (GP)-based approach to automatically create multiple new r.fs for each original (*source*) feature [8]. The fitness function was designed to maximise the redundancy between the source and created features, and minimise the redundancy between created features, measured using mutual information (MI), so as to produce varying types of r.fs. While this work was able to produce significantly more complex (non-linear) feature redundancies than the naïve ones described above, these redundancies were still univariate. In addition, minimising the redundancy between a set of created features (with the same source feature) is not ideal, as it


*GECCO '18, July 15–19, 2018, Kyoto, Japan*

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205552

does not directly measure how different their redundancy relationships with the source feature are. Such an approach can not be used when creating multivariately redundant features, as it contradicts with the aim of producing complex multivariate relationships.

This work aims to propose a more refined and advanced GP method to evolve sets of r.fs which are multivariately redundant to the original features of the dataset. Specifically, we aim to:

- Develop a multi-tree GP representation to allow multiple source features to be used by a GP individual in each tree.
- Propose a suitable and efficient fitness function that allows difficult, complex multivariately redundant features to be automatically created.
- Analyse the performance of feature ranking and feature selection techniques on the created features to explore how they affect the performance of these algorithms.
- Compare with the existing univariate GP approach and discuss the differences in the features created by the existing and proposed method.

It is expected that this work will give more challenging and robust datasets that can be used for benchmarks feature selection algorithms.

## 2 BACKGROUND

This section will briefly touch on important concepts of feature manipulation and mutual information, before summarising the key points of the existing work that this paper builds on.

### 2.1 Feature Manipulation

Feature manipulation algorithms actively change the feature set of a dataset in order to improve the results of a machine learning task. Feature selection (FS), 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, from simpler feature ranking [12] and sequential forward/backward search techniques [16, 19] to more advanced methods using sparse models [1, 2] or evolutionary computation (EC) search techniques [20].

Feature construction (FC) techniques take a different approach, whereby they attempt to construct new, higher-level features which are transformations of multiple features in the dataset. The new constructed features are trained to have improved performance, while also reducing the number of features used by the machine learning algorithm. The most popular EC-based FC method is GP [4, 14, 18], while common non-EC methods such as Principal Component Analysis (PCA) [6] and Support Vector Machines (SVMs) [3] are also often considered to perform a form of FC. GP-based FC has a close relationship with the current work, as feature construction shares many similarities to redundant feature creation.

### 2.2 Mutual Information

Mutual Information (MI) is one of the most popular measures of the redundancy between two variables, as it essentially measures the amount of information that two variables *share*. Hence, the higher MI between two features, the more redundancy they can be said to share. The MI between two features $X$ and $Y$, $MI(X, Y)$, is defined as follows [5]:

$$MI(X, Y) = H(X) + H(Y) - H(X, Y) \tag{1}$$

where feature $X$ has entropy $H(X)$ and two features $X, Y$ have joint entropy $H(X, Y)$, defined in the standard way [5]. For continuous features, such as in this work, the full MI equation is as follows:

$$MI(X, Y) = \int_X \int_Y p(x, y) \times \log_2 \frac{p(x, y)}{p(x)(y)} dx\ dy \tag{2}$$

As this work uses the multivariate form of MI, where there are multiple source and target features, an extension of Equation (2) is used, where $X$ and $Y$ represent the set of source and target features respectively, e.g. $\mathbf{X}$ and $\mathbf{Y}$. 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.

Equation (2) requires knowing the full marginal and joint probability density functions (*pdf*) of the features; in practice, this is not available as each feature represents only a sample of the underlying *pdf*. To remedy this, a common technique is to use an estimation algorithm to estimate MI. One of the most successful estimators is the nearest-neighbour estimation approach proposed by Kraskov et al. [7], which uses the similarity of the nearest neighbours for each instance (for the given features) as a proxy for measuring the strength of the redundancy between the two features. The Java Information Dynamics Toolkit (JIDT) [13] implementation of this algorithm is used in this work.

### 2.3 Related Work: GPRFC

As this is only the second known work on using EC to automatically created r.fs from a source dataset, our focus in this section is on the previously proposed method (GPRFC) by Lensen et al. [9] as it forms the baseline and inspiration of this work.

Genetic Programming for Redundant Feature Creation (GPRFC) [9] uses a multi-tree GP representation, where each GP individual contains $n$ trees. Each tree in an individual represents a single mapping from a source feature to a constructed redundant feature. Hence, a new feature is generated iteratively by setting the input of the tree as the source feature, evaluating the tree from bottom to top, and taking the output as the corresponding feature value of the new created feature. By having a single individual contain multiple trees, the authors were able to design a fitness function that encouraged each tree to represent a **different** mapping, while still ensuring that each mapping still produced a new feature that was highly redundant with the source feature. A range of functions were included in the function set to allow for a range of differently shaped mappings between the source and created features, including arithmetic, trigonometric, boolean, and conditional operators which allow for non-continuous mappings to be formed.

GPRFC used a fitness function which used mutual information (MI) as a proxy for measuring the redundancy between features. As such, their proposed fitness function sought to maximise the MI between the source and created feature, while minimising the MI between all pairs of created features. They model this by attempting to maximise the *minSourceMI* (the minimum MI between the source feature and any created feature) and minimising the *maxSharedMI* (the maximum MI between any pair of created features). The authors also considered that when *minSourceMI* falls below a certain threshold, the created r.fs are not acceptably redundant and so are infeasible solutions: in this case, the fitness function considers only

the source MI in order to prioritise producing feasible solutions at the start of the GP run.

GPRFC also used several other "tricks" to optimise the GP performance, including scaling each source and created feature to a pre-defined range and adding a small amount of input noise to better manage duplicate source feature values. We refer to the original paper for further discussion on these details. It is important to note that GPRFC required running a single run of GP for each source feature in a dataset, which makes it costly for large feature sets. In contrast, a proposed multivariate approach will use multiple source features per run, requiring less computational time.

*2.3.1 Summary.* While GPRFC was a promising first attempt at using GP to automatically create r.fs, it has a number of limitations which this work seeks to address. It was only designed to create univariate r.fs, which are inherently less difficult to identify than multivariate r.fs. The fitness function minimised the MI between created features, which does not directly measure the difference in the mapping between each source and created feature. 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. 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. This work seeks to address these limitations.

## 3 PROPOSED METHOD: GPMVRFC

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:

$$|\mathbf{X}| = \lfloor \max\left(2, \min\left(\frac{m}{4}, 5\right)\right) \rfloor \quad (3)$$

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 use all of the source features, and hence different trees may be redundant with different subsets of the source features.

### 3.1 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 1. Note that the cosine, subtraction, and division operators are not included as they are complements of included operators, and were found to decrease performance.

**Table 1: 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$(a, b, c)$ [1] | 3 |

### 3.2 Fitness Function

The fitness function used in this work consists of two components: one which measures the redundancy between the source feature set and the created feature set using multivariate MI, and another which 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:

$$\text{Redundancy} = MI(\mathbf{X}, \mathbf{Y}) \text{ as per Equation (2)} \quad (4)$$

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 earlier, an estimator is used to compute the approximate MI as the unknown *pdf* 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. Fig. 1 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 different shapes in the feature space, and so represent different types of r.fs. The two linear functions,

[1] If $a$ is non-negative, output $b$; otherwise, output $c$.

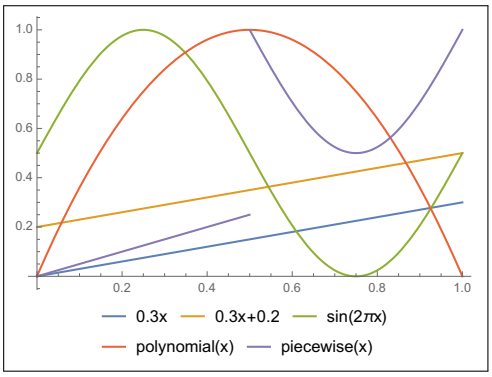



**Figure 1: Example of some (scaled) redundancy mappings that could be created by GPMVRFC.**

however, both are the exact same type of r.f, except that they have different offsets from the x-axis. The piecewise and sin functions have the same redundancy function for half 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 which 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 1. We take the minimum of these differences as the minimum difference between $\mathbf{A}$ and any other created feature, called $\text{RFDiff}_{\text{min}\mathbf{A}}$. In addition, we also consider the difference between $\mathbf{A}$ and each of the source features (using Algorithm 1), to ensure that no created feature inadvertently reconstructs one of the existing features (i.e. making it naïvely redundant). The minimum difference between $\mathbf{A}$ and any source feature $x \in X$ is computed as $\text{SourceDiff}_{\text{min}\mathbf{A}}$. The smaller of $\text{RFDiff}_{\text{min}\mathbf{A}}$ and $\text{SourceDiff}_{\text{min}\mathbf{A}}$ is the distance between $\mathbf{A}$ and any other constructed feature or source feature. The final formulation of the difference of the created r.fs is as follows:

$$\text{Difference}_{\text{RFs}} = \min_{\mathbf{A} \in Y} \min\{\text{RFDiff}_{\text{min}\mathbf{A}}, \ \text{SourceDiff}_{\text{min}\mathbf{A}}\} \quad (5)$$

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

$$\text{Fitness} = \text{Redundancy} \times \text{Difference}_{RFs} \quad (6)$$

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.

**Algorithm 1:** Computing the difference of two created feature vectors, $\mathbf{A}$ and $\mathbf{B}$, given source features $\mathbf{X}$.

$Difference_{min}$ = 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 $\frac{1}{|x|} \times Gradient_{diff}$

## 3.3 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 briefly summarised below, and further discussion can be found in the original paper [9].

- In order to ensure all source features have the same range, we scale each source feature to fall in the range $[0 + \epsilon, 1 + \epsilon]$ where $\epsilon$ is a small weighting added to prevent 0 values from making invalid/poor trees as often. We set $\epsilon = 1 \times 10^{-3}$.
- Duplicate feature values are problematic as they reduce the number of unique inputs to a GP tree, and hence the ability of the GP to fine-tune solutions. This is remedied by adding a small amount of random noise (with a fixed seed), called $\delta$ to each source input in a tree. We use the same approach as in GPRFC, where $\delta$ is randomly sampled from $[0.001\epsilon, \epsilon]$. Note that this, and the above tweak, are used only for producing the output of a tree — not for measuring fitness.
- The created features are also scaled to $[0, 1]$ to ease interpretability, and so that they have the same range as the source feature — useful for algorithms which are distance-based.

## 3.4 GP Parameters

We used a population size of 1,024, 200 generations of evolution, 40% mutation, 60% crossover, top-10 elitism and a max depth of 15. These parameter settings are similar to those in GPRFC in order to maintain a fair comparison and hone the focus of this paper. 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.

# 4 EXPERIMENT DESIGN

To evaluate the performance of the 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 [10]. The number of source

**Table 2: 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 |

features and the number of features in the datasets produced by GPMVRFC, and GPRFC, are shown in Table 2. 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) [5, 12]. 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 (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") [22] to perform FS (i.e. sparse feature selection), as well a Joint Mutual Information (JMI)-based search [21].

*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 Search (SFS), Sequential Backward Search (SBS), and the floating versions of SFS (SFFS) and SBS (SFBS) [17] 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 [15].

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 RESULTS AND DISCUSSION

Due to the number of different test combinations across the seven datasets, it is impractical to show and analyse all results in an interpretable way within the page limit. Hence, we instead focus on analysing the results of the four datasets which 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.

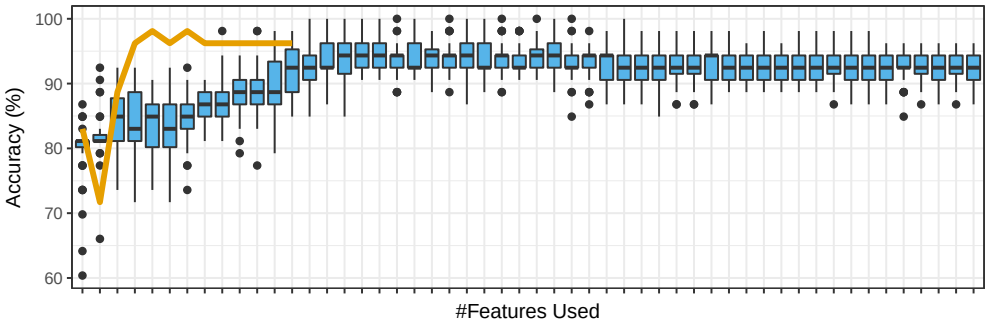
### 5.1 Wine

Fig. 2 shows the results of the FS methods on each of the 40 augmented Wine datasets. Fig. 2a 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 red 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 which 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 4 features on the original dataset would give.

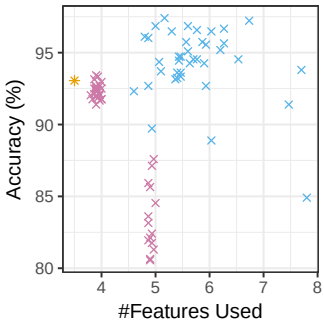
The remainder of the plots show one FS approach for each of the 40 augmented datasets for GPMVRFC and GPRFC, and the original dataset. A small amount of noise is added so that duplicate points can be distinguished. The sequential FS algorithms (SFFS and SFBS; Figs. 2c,2d) show very similar accuracy distributions 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 (Fig. 2e) 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 (Fig. 2f) 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 (Fig. 2b) is particularly interesting, as it shows GPMVRFC actually improving the performance of the classifier, albeit with more 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.

### 5.2 Dermatology

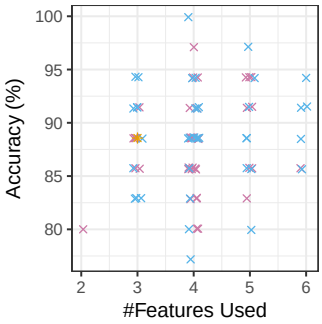
The results on the Dermatology dataset (Fig. 3) further reinforces the hypothesis that GPMVRFC may actually be inadvertently creating more meaningful r.fs. Feature ranking (Fig. 3a) shows that selecting the same number of features on the augmented datasets can give *better* test accuracy than on the original dataset for the first $\approx 15$ features. This pattern continues on four of the remaining FS methods (Figs. 3c–3f) where GPMVRFC datasets generally have higher accuracy compared to GPRFC, and often even compared



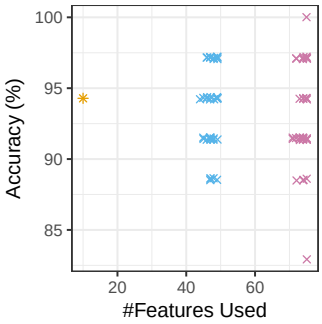

(a) Features ranked by IG, classified with KNN.



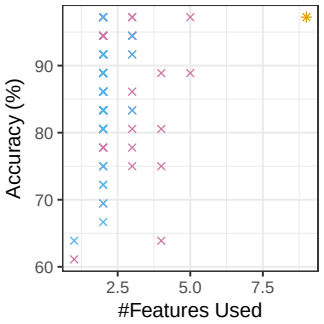

(b) Classified with DT.



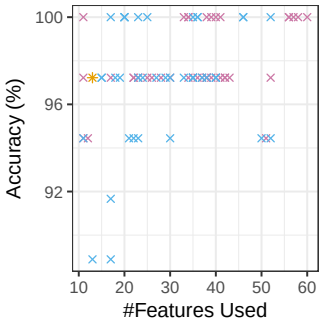

(c) FS with SFFS, classified with KNN.




(d) FS with SFBS, classified with KNN.

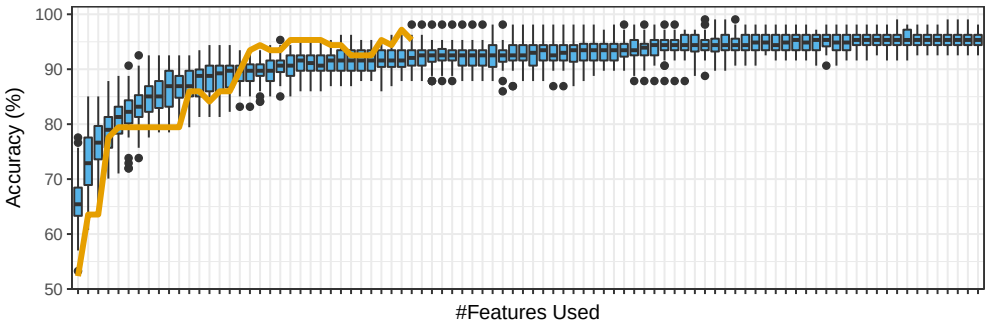



(e) FS with l1SVC, classified with KNN.

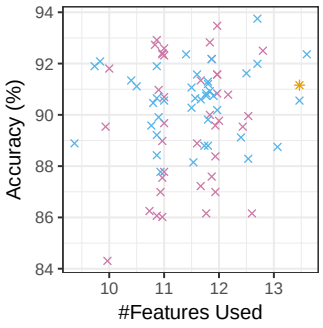



(f) FS with JMI, classified with KNN.

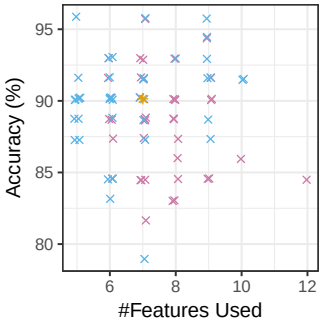
**Figure 2: Wine Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is existing GPRFC method.**



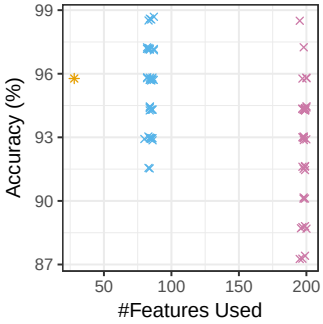

(a) Features ranked by IG, classified with KNN.



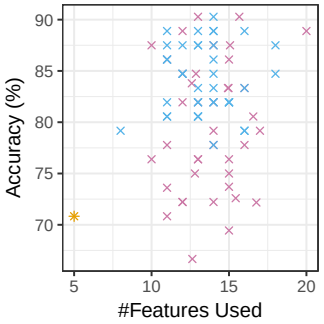

(b) Classified with DT.



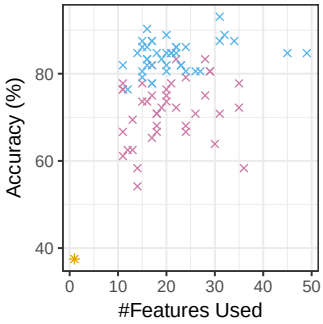

(c) FS with SFFS, classified with KNN.




(d) FS with SFBS, classified with KNN.

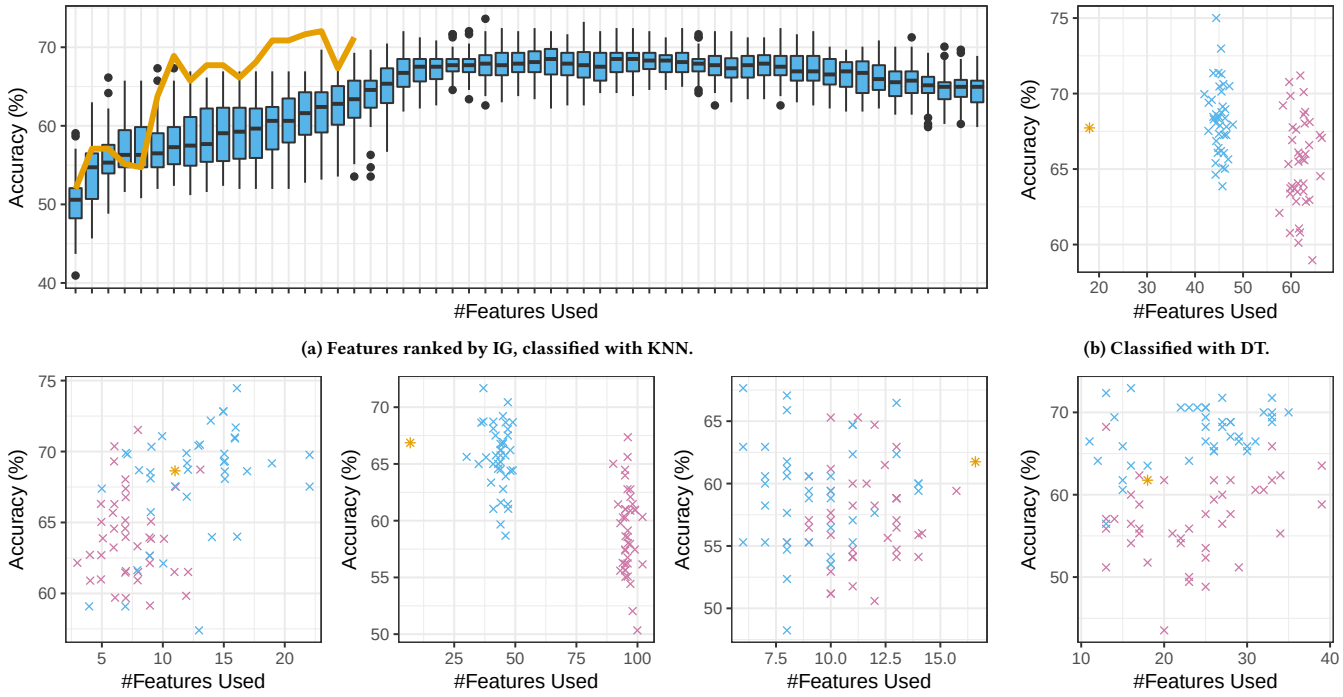



(e) FS with l1SVC, classified with KNN.




(f) FS with JMI, classified with KNN.

**Figure 3: Dermatology Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is 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 4: Vehicle Dataset. Orange is the original dataset; blue is GPMVRFC; and pink is existing GPRFC method.**

to the original dataset. In the case of SFFS, GPMVRFC often gives higher accuracy with fewer features, and 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 (Fig. 3b) 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.

## 5.3 Vehicle

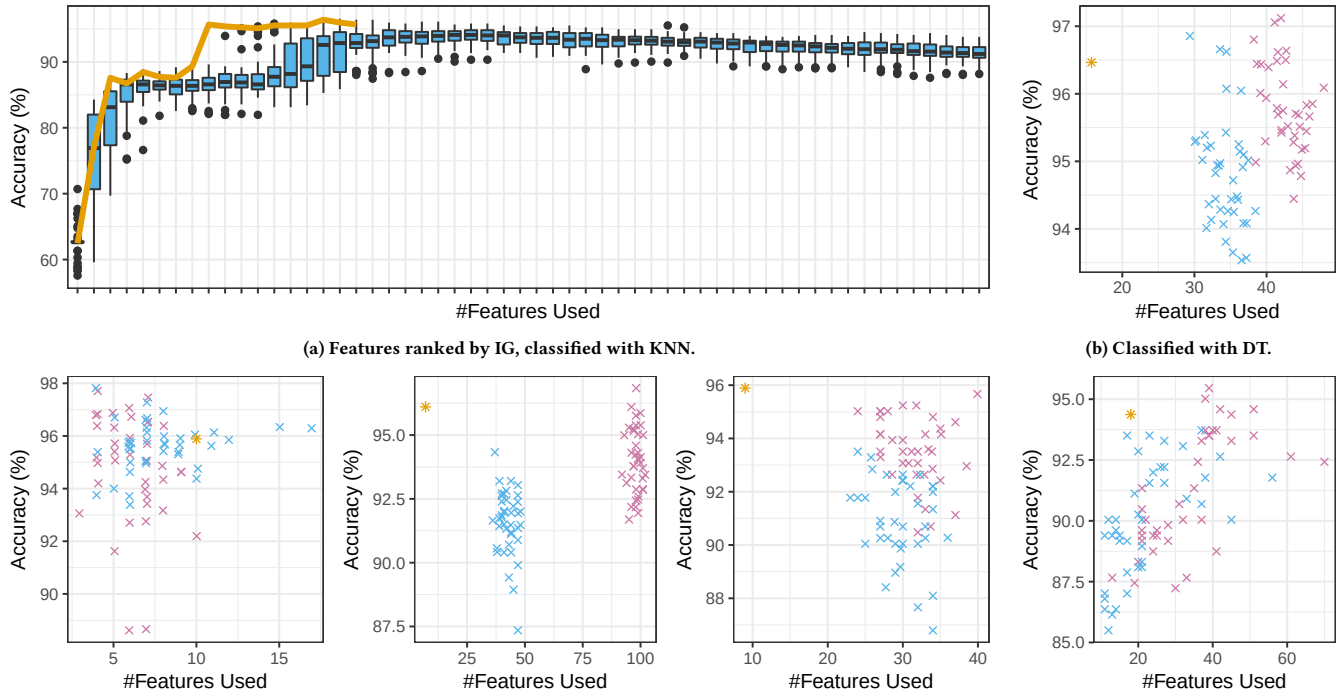
The Vehicle dataset results (Fig. 4) continue to show a similar pattern. The primary exception is feature ranking (Fig. 4a), where the GPMVRFC-augmented datasets clearly decrease the performance of the classifier. As discussed before, this may be due to feature ranking being a univariate approach that cannot properly cope with multivariate feature interactions. Interestingly, the r.fs also cause the performance of the classifier *decrease* when the last third of features are used, suggesting some features may be misleading to the KNN classifier. Each of the remaining results (Figs. 4b–4f) 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.

## 5.4 Image Segmentation

On the final dataset, the general pattern in the results (Fig. 5) is that GPMVRFC produces r.fs that cause **lower** test accuracy than GPRFC, despite GPRFC producing twice as many r.fs. Feature ranking clearly suffers in accuracy, including a slow drop off in accuracy even after only half the best-ranked features have been used (Fig. 5a). The SFBS, l1SVC, and DT methods (Figs. 5d,5e,5b) 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 (Fig. 5f) under-selects features compared to the original dataset, or selects extra, less useful features; SFFS (Fig. 5c) is the exception to the pattern in that it doesn't have clearly worse accuracy due to GPMVRFC, but it also selecting many fewer features than the other FS methods and so may give less over-fitting. We plan to investigate why GPMVRFC gave significantly different results on the Image Segmentation further in the future.

# 6 CONCLUSION

This work aimed to develop a new method for automatically creating multivariately redundant features in order to make difficult benchmark feature selection datasets. We proposed a new GP representation, and a more appropriate and efficient fitness function compared to existing work. 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




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

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. We will investigate this phenomenon further in the future in order to explore how unsupervised feature creation can improve supervised learning performance, as well as investigating adversarial methods for creating multivariate redundant features that can directly cause the performance of common feature selection algorithms to significantly suffer.

## REFERENCES


[1] Jinbo Bi, Kristin P. Bennett, Mark J. Embrechts, Curt M. Breneman, and Minghu Song. 2003. Dimensionality Reduction via Sparse Support Vector Machines. *Journal of Machine Learning Research* 3 (2003), 1229–1243.

[2] Paul S. Bradley and Olvi L. Mangasarian. 1998. 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*. 82–90.

[3] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[4] Pedro G. Espejo, Sebastián Ventura, and Francisco Herrera. 2010. A Survey on the Application of Genetic Programming to Classification. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 40, 2 (2010), 121–144.

[5] Edwin T Jaynes. 1957. Information theory and statistical mechanics. *Physical review* 106, 4 (1957), 620.

[6] Ian T. Jolliffe. 2011. Principal Component Analysis. In *International Encyclopedia of Statistical Science*. Springer, 1094–1096.

[7] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. 2004. Estimating mutual information. *Physical review E* 69, 6 (2004), 066138.

[8] Andrew Lensen, Bing Xue, and Mengjie Zhang. 2018. Generating Redundant Features with Unsupervised Multi-tree Genetic Programming. In *Genetic Programming - 21st European Conference, EuroGP 2018, Parma, Italy, April 4-6, 2018, Proceedings (Lecture Notes in Computer Science)*, Vol. 10781. Springer, 84–100.

[9] Andrew. Lensen, Bing. Xue, and Mengjie. Zhang. 2018. Generating Redundant Features with Unsupervised Multi-Tree Genetic Programming. *ArXiv e-prints* (2018). arXiv:1802.00554

[10] M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml

[11] Huan Liu and Hiroshi Motoda. 2012. *Feature selection for knowledge discovery and data mining*. Vol. 454. Springer Science & Business Media.

[12] Huan Liu and Zheng Zhao. 2009. Manipulating Data and Dimension Reduction Methods: Feature Selection. In *Encyclopedia of Complexity and Systems Science*. 5348–5359.

[13] Joseph Troy Lizier. 2014. JIDT: An Information-Theoretic Toolkit for Studying the Dynamics of Complex Systems. *Front. Robotics and AI* 2014 (2014).

[14] Kourosh Neshatian, Mengjie Zhang, and Peter Andreae. 2012. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Trans. Evolutionary Computation* 16, 5 (2012), 645–661.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[16] Pavel Pudil, Jana Novovicová, and Josef Kittler. 1994. Floating search methods in feature selection. *Pattern Recognition Letters* 15, 10 (1994), 1119–1125.

[17] Jiliang Tang, Salem Alelyani, and Huan Liu. 2014. Feature Selection for Classification: A Review. In *Data Classification: Algorithms and Applications*. CRC Press, 37–64.

[18] Binh Tran, Bing Xue, and Mengjie Zhang. 2016. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing* 8, 1 (2016), 3–15.

[19] A. Wayne Whitney. 1971. A Direct Method of Nonparametric Measurement Selection. *IEEE Trans. Computers* 20, 9 (1971), 1100–1103.

[20] Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. 2016. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Trans. Evolutionary Computation* 20, 4 (2016), 606–626. https://doi.org/10.1109/TEVC.2015.2504420

[21] H Yang and John Moody. 1999. Feature selection based on joint mutual information. In *Proceedings of Computational Intelligence Methods and Applications (CIMA), New York, USA*. 22–25.

[22] Ji Zhu, Saharon Rosset, Trevor Hastie, and Robert Tibshirani. 2003. 1-norm Support Vector Machines. In *Advances in Neural Information Processing Systems (NIPS) 16, December 8-13, 2003, Vancouver, Canada]*. 49–56.