**Response to Reviewers**

As was pointed out by the reviews for this work, a greater focus has now been presented towards big data and how this work relates to big data solutions more. For both the filter (chi-squared feature ranking) and the embedded (random forest feature ranking) methods for feature selection, a more comprehensive evaluation has been performed on how each method has been implemented in terms of big data architecture design. This has included analysing and considering the map and reduce stages of each of these methods, and identifying if they are global or local models. This has been particularly the case for the random forest element of the work, found to be a local method, where a specific emphasis has been placed on how the random forest architecture is partitioned, how parallelism has been considered, how mapping has been performed, and how the results and predictions of each partition have been aggregated and reduced. Additionally, specific attention has been paid towards random forest subsampling here as has been suggested, including being explicit on repartitioning using Spark, shuffling of the dataset and picking samples with equal class representation. Furthermore, the divide-and-conquer approach presented by the random forest method has been discussed, and its advantages to big data applications have also been noted.

In addition, the background on filter methods in general has been more intimately explored, which has included looking at not only the advantages of filter methods in the traditional sense, but also in how filter methods can be popular when dealing with big data, such as due to their scalability which is highly relevant to its application in big data and, thus, the rest of this work. In addition, greater care has been taken in justifying the use of chi-square for the filter method in big data such as with its robustness with respect to the distribution of data and ease of computation, as well as how it has been implemented for big data using PySpark, and how the map and reduce steps have been approached. The chi-square methodology was also identified to be a global method.

For both chi-square and random forest methods, the limitations of each have also been more specifically discussed, such as by making particular note towards the effect that noisy data can have on feature selection methods, particularly in the case of random forest which can be very negatively affected by noisy data. Finally, in terms of the filter and embedded methods considered, less of the material now consists of the history of such methods which was the case beforehand, now instead the work looks more exclusively at the knowledge that is relevant to this work, such as how the methods work, what is involved, and their benefits and limitations.

The dataset utilised here has been more explicitly described in terms of what it is, how it has been artificially generated for this work, and why it has been generated in this way (as opposed to using a real-world dataset). In terms of how the different methods of feature selection perform using this dataset, measurements have been more clearly expressed for each method, including the feature selection accuracy, including which features, whether useful or useless, have been selected by each method, and the time taken to perform each method.

It was noted in at least one of the reviews that the work being undertaken here is not novel and that it doesn’t look to fill a gap in wider research. Whilst there are countless papers that exist which analyses, compares, and contrasts filter and embedded methods, there are few that do so in the frame of big data. It would be interesting if the reviewers could have utilised some references and previous literature to back up the notion that this work is not novel, but insofar as the reviews produced for this work conclude, there have been none to base this on. The gap in the research that has been utilised in this work has now been emphasised so as to make sure that the novel purpose of the resulting paper is clear.

Finally, as suggested by the reviewers, the formatting of the work has been tidied up to conform with IEEE standards and style guidelines. This has included making sure all captions are appropriately applied to tables, equations, and figures, as well as making sure references are appropriately referenced and recorded. Additionally, more information and details with regard to the work to be performed and its findings have been included in the introduction of the paper, and an abstract has been added that adequately and briefly reflects what the paper presents. Furthermore, more visual elements have been included such as diagrams to make the architectures and structures of different components of this work more obvious and clearer, which helps to reinforce what is explained in the text. With regard to how these visual elements are presented, the appendix has been removed and all diagrams are now present in the main body of the paper.

A Comparison of Embedded and Filter Methods for Feature Selection in Big Data

Chiamaka Anamekwe  
*School of Computer Science*   
*University of Nottingham*Nottingham, UK  
psyca@nottingham.ac.uk

.Kimiya Sagha  
*School of Computer Science*   
*University of Nottingham*Nottingham, UK  
psyks10@nottingham.ac.uk

Joshua McDonagh  
*School of Computer Science*   
*University of Nottingham*Nottingham, UK  
psyjm10@nottingham.ac.uk Raul Ghisa  
*School of Computer Science*   
*University of Nottingham*Nottingham, UK  
psxrg6@nottingham.ac.uk

*Abstract*—The curse of dimensionality is a big issue that arises often in big data. Having too many features makes training models harder and more costly, and methods to deal with this is a heavily researched area. Feature selection is one such method that aims to reduce the dimensionality of data, and is the focus of this paper. In current literature, most solutions focus on filter methods to achieve this due to their speed, with less focus on other approaches, and thus the aim of this paper is to explore an alternative, embedded methods, and determine whether they could provide a viable option.

In this work, we implement a random forest as our chosen embedded method, adopting a divide-and-conquer approach in order to parallelise the method and scale it up to deal with big data, and we implement the chi-squared test as our chosen filter method to compare against. Using an artificial dataset, we compare the selected features of both models against that of the known useful features. We find that both models select the correct features with a similar accuracy, with random forests performing marginally better. Following with other literature, we find that the chi-squared test performs much faster, although when including the discretisation of our continuous data, it had a higher computational time.

Based on our results, we conclude that embedded methods can be a viable alternative to filter methods in some cases, and thus more research is needed in the area.

# Introduction

As datasets grow larger and the amount of data available continues to increase exponentially, it becomes increasingly difficult to deal with it. Filter methods are the current industry standard used for feature selection to deal with high dimensional data as they are found to be faster, easier to implement, and more suited to commercial use, although embedded methods tend to have a greater accuracy.

This study aims to explore the use of an embedded method for feature selection in the context of big data. Unlike other methods, such as filter and wrapper which perform feature selection as a pre-processing step, embedded methods perform this as part of the training process [1]. In this study, the use of an embedded method for feature selection in the context of big data is explored. It is important that these methods are adapted to work in a big data context, as a normal implementation would have a poor efficiency, defeating the point of performing feature selection to ultimately improve efficiency. For this study, the embedded method will be implemented in a random forest (RF) which can be parallelised with a divide and conquer approach for scalability, making it more suited to deal with large datasets. To better understand its performance, it will be compared to a filter method. To compare the two feature selection algorithms, a support vector machine (SVM) will be trained with the derived subset of features from both the chi-squared method and the RF model, and their performance will be measured against that of an SVM trained with the complete dataset (useless features included), and an SVM trained with a dataset of only the known useful features.

In conclusion, the goal of this study is to investigate whether an embedded method could be equally or more effective for feature selection scaled up to big data, compared to a filter method.

The paper is structured as follows. Section II provides background literature on feature selection, filter methods and embedded methods. In Section III, the implementation of RF is discussed, and the methods adopted to scale it to big data. The implementation of the chi-squared test is also discussed. Section IV describes the datasets used and explains the performance metrics used in the testing and then provides detail on the experiments performed. In Section V the results of the experiments are presented and discussed, and Section VI summarises and concludes the paper.

# Background

## Feature Selection

A major problem in the utilisation of machine learning techniques is that of how datasets with high dimensionality are handled. Highly dimensional datasets will likely contain features that are noisy or redundant, which can be particularly relevant in the case of big data as such datasets will often include unwanted, incomplete, or redundant features [2]. Datasets, particularly those of big data, can thus be especially inefficient for machine learning models to train on, and potentially make such models poorer performing and badly generalised, as is found in [3]. As a result, in such cases where datasets with high dimensionality and redundant features are being handled, it is important to consider some form of dimensionality reduction, such as feature selection [4], to reduce the dimensionality of the problem instance at hand.

Feature selection is the process of selecting the features of a dataset that are most predictive and informative when it comes to the associated class labels. Feature selection is widely used in a diverse range of problems, such as to assist with Twitter sentiment classification [5], emotion recognition [6], and, more relevantly, distributed big data processing [7]. How feature selection achieves this dimensionality reduction can vary; as is discussed in [8], feature selection can be achieved via three different methods: filter, wrapper, and embedded. This work focuses on the filter and embedded methods.

## Filter Methods

An approach to feature selection is that of the filter method, which operates independently of the machine learning algorithm and operates very much as a pre-processing step [9], as is shown in Fig.1. As observed by [10], generally, filter methods consist of two steps: the first step is to have the features ranked by criterion including informativeness and dependency [11], and the second step is to select the highest-ranking features of those returned by the first step. The result of this is a dataset with a reduced dimensionality consisting only of the most informative features, which can then be utilised for further machine learning processing. Note that using exclusively informative features is useful as it helps focus the machine learning algorithm on the features of the dataset that are most representative of the underlying patterns that can be found.

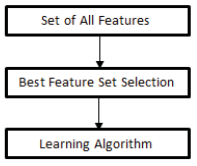


Figure 1: Filter method for feature reduction as given by [10].

Not only are filter methods simple to implement, but they are often reliable [10], and avoid overfitting [11]. Additionally, as is remarked by [11], filter methods are easily scalable to larger datasets and datasets with a particularly large dimensionality, and are notable for being computationally inexpensive and requiring less computational time, which is relevant and important when considering the application of feature selection to big data. As is noted by [12], this is reflected in how commonly filter methods are utilised for big data, with examples including the classification of patient records [13], and the ‘internet of e-health things’ using big data [14].

## Embedded Methods

Another approach to feature selection is that of the embedded method, which performs feature selection during the machine learning algorithm’s training stage, and, thus, is specific to the machine learning algorithms involved [15]. An important quality of embedded feature selection, therefore, is that the processes of feature selection and machine learning are not separate, and that the selection of the most informative features and the training of the learning model occurs simultaneously [16].

Embedded feature selection is useful because its feature selection process is guided by the performance and training of the learning algorithm (unlike the filter method), but does so using relatively little computational cost, unlike one of the other alternative feature selection processes: the wrapper method [17]. This is particularly relevant to the application of feature selection in big data, as a reduced computational cost can be highly beneficial. As a result of this advantage, the embedded method of feature selection will sometimes be used to handle big data problems, such as for big data feature selection for SVMs [18], and for traffic classification schemes [19].

# Methodology

## Random Forest Feature Ranking

For the embedded method, the RF classifier algorithm will be used, which is based on an ensemble of decision trees. During the training phase, multiple datasets are created from random samples of the original dataset and the decision trees are trained on them, with the number of decision trees being decided based on trial-and-error tests.

The high parallelisation potential of the algorithm is exploited in big data using the divide-and-conquer approach. After the dataset is split, each worker is mapped to a given subset and builds a decision tree in parallel – see Figure 2 for a visualisation of this where *Q* is the number of decision trees trained in parallel. In the reduce stage, all the models are aggregated in a RF and the final prediction in a classification problem is the majority vote of the trees in the ensemble. As each tree sees only a fraction of the original dataset, the divide-and-conquer approach is a *local* big data solution.

Diagram

Description automatically generated

Figure 2: Random Forest divide-and-conquer approach [20]

This approach greatly decreases the computational time, as most of the workload is done in parallel before being aggregated into the final RF. It also minimises the need for data movement across worker nodes, as each forest only needs to be trained on the data it has access to.

An important side-output of the RF is feature importance ranking. Each decision tree tries to optimise the node split by maximising the impurity gain. The amount of impurity gain gives an insight about how important a certain attribute is in predicting the output. We can then consider each feature’s impact on the impurity over all the trees in the forest and compute the features’ ranking based on their importance. As these values are already computed in the training phase, this method is extremely quick and can also be parallelised [21]. The ranked list is a good indication of which features provide the most useful information and which features can be discarded.

Methods like neural networks or support vector machines are complicated algorithms, and they are not understood well enough to give insights on how a certain feature impacts the prediction [22]. Ensemble methods such as RF perform better than single classifiers when it comes to feature selection because they overcome bias in the data and avoid local optima by combining the results of multiple uncorrelated classifiers which cancel out each other’s bias [23].

As discussed in the previous paragraphs, the impurity function plays a significant role in feature selection. The Gini impurity is a probabilistic measure of the purity of a dataset and is frequently used in RF [24], where a value of 1 signifies a pure node. Another impurity function is the information gain that approaches the purity from the point of view of entropy and a pure node carries 0 information gain. According to some studies, Gini impurity and information gain behave similarly in experiments, but Gini is preferred as it does not involve logarithmic operations, making it faster [25]. The feature importance in RF is then measured by summing the Gini impurity gains (), for each feature, in each node of every tree [26] [27] (1).

where *I* is the feature importance, *f* is the feature, *n* is a node in the tree, *T* and is the Gini impurity (2).

where *P* is the fraction of the training examples that reach the node *n* with the label .

The RF function available in PySpark, i.e., *RandomForestClassifier*, is used to train and evaluate the embedded method. The algorithm implements the divide-and-conquer approach with Gini impurity; thus, the training time is optimised for big data. The algorithm returns a RF model, and the feature importance ranking is accessed through the *featureImportances* property.

## Chi-Squared Feature Ranking

The chi-squared test will be used to implement the filter method of feature selection. It is a statistical method which can be utilised for the ranking of features, as it allows for the estimation of a class label’s independence of a feature [28]. As is stated in [29], the chi-squared test, with respect to the classes, evaluates each feature individually. For this statistical method, continuous valued features must be discretized and represented using discrete quantities in the form of intervals. With this, the chi-squared test compares the frequency of a class for each interval with the expected class frequency. As a result, the chi-squared value can be calculated

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

where is the number of class labels, is the number of intervals, is the number of examples, is the number of samples of the class within the th interval, and is the expected frequency of which is where is the number of samples in the th interval [29]. With this statistic, we can then determine how informative a feature is by how great the value is: a greater value means the feature is more informative. This statistic will then inform which features to select and which features to discard.

The reason why the chi-squared test is being considered here is because it has a good track record of producing classifiers with high fitness, such as in [30] where the chi-squared test produces a classifier with higher accuracy than the information gain alternative for all different numbers of features tested (100, 400, 700, 1000, 1300, 1500, 2000). Additionally, [28] compares the chi-squared test against a greedy algorithm (Greedy), as well as information gain ratio (IG-Ratio), and Minimum Redundancy and Maximum Relevance (MRMR). Here, all four methods are tested using four different classification models: decision-tree, naïve Bayes, k-nearest neighbour, and an artificial neural network. As the results of this found by [28], the accuracy for each model using the chi-squared test is decent, particularly for the decision-tree classifier. Thus, given chi-square’s advantages, and the benefits of the filter method more widely, chi-square is utilised commonly in big data solutions, such as for stroke prediction [31] for example, where the use of chi-square also enables high accuracy.

In terms of its implementation in this study, discretisation has been incorporated in the Python code as a way of pre-processing the data to enable the chi-square calculations. It is done using the *QuantileDiscretizer* function, supplied by the DataFrame-based PySpark library MLlib, which is optimised for parallelised computing [32] [33]. This function converts a column of the big data dataset with continuous values into a column of discrete values, depending on a pre-defined number of quantiles (set as ‘*numBuckets’*) based on the distribution of the data - thereby eliminating the need to manually calculate meaningful splits. The most optimal value of numBuckets will be found through trial-and-error testing. The *QuantileDiscretizer* is iteratively executed for each continuous valued column of the dataset and, as it has been implemented in Spark, the function discretizes the continuous data in a distributed way.

The global method function that is then used is that of *ChiSquareTest.test*, which is also supplied by PySpark [34]. This function implements the chi-squared test such that it is able to run on distributed partitions of the data. In order to achieve this, the function takes in the dataset as a DataFrame, as well as the names for the feature columns, and the class label column name. This function then maps the chi-squared test calculation over all the dataset features, and reduces the result into a DataFrame which holds the feature rankings found for each feature. This resulting DataFrame is returned and can then be used to finally select the features that are most useful and informative.

The system proposed for the feature selection evaluation is described in Figure 3. For different artificial datasets, both the RF model and the Chi-squared test will be used to generate the feature ranking. Knowing the features that are relevant in advance, the feature selection accuracy will be measured. To have a real-world performance measure for the selected features, a linear support vector classifier (linear SVC) will be trained on the reduced dataset and the obtained classification accuracy is further interpreted. The time taken for the RF and the Chi-squared algorithms to compute the feature ranking is also stored and analysed from a big data scalability perspective.

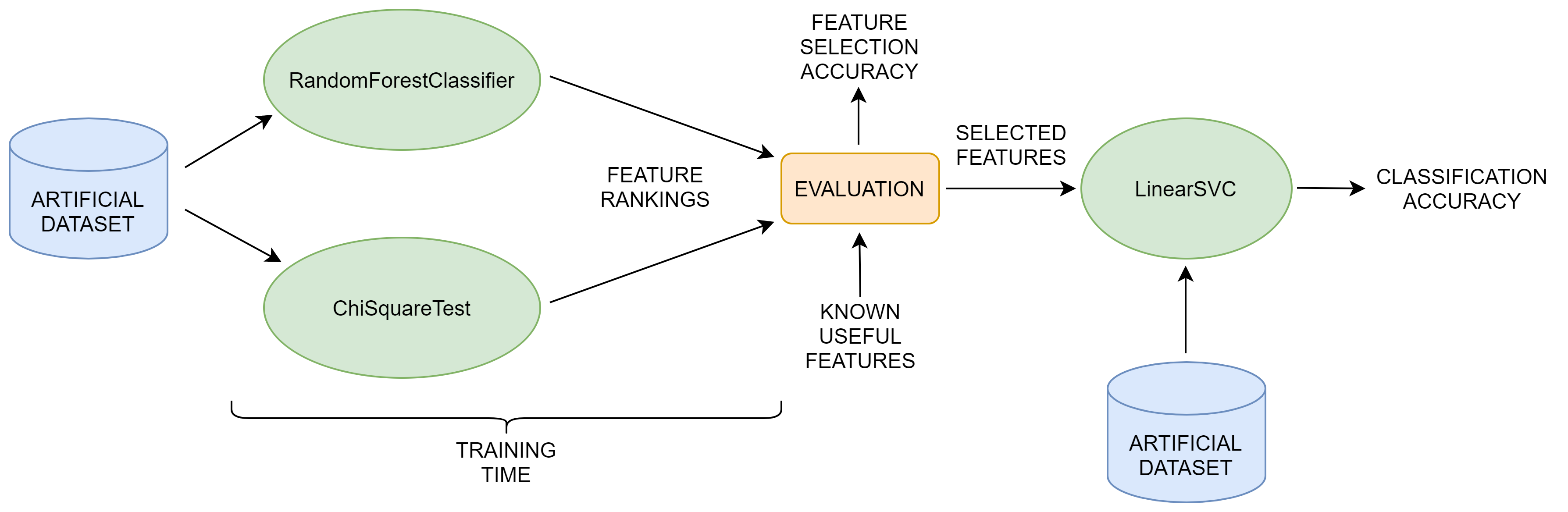


Figure 3: Proposed system architecture

# Experimental Setup

## Datasets

The properties of the datasets highly influence the performance of the feature selection algorithms. Studies have shown that decision trees are especially sensitive to feature noise [35]. Their feature selection accuracy decreases as the ratio of the features that are not correlated with the label increases. In order to validate the feature selection methods, both real and artificial datasets were initially considered. Although the feature selection application on real datasets gives a more realistic overview of the real-world performance, the important features are not known in advance and the feature selection method cannot be objectively evaluated. To have an unbiased evaluation of the feature selection methods, this research uses artificial datasets where the features that are correlated with the label (or useful features) are known in advance. Noise is then added by generating random features (or useless features) that are uncorrelated with the label.

The artificial datasets were created with the algorithm used to generate samples for feature selection competitions[[1]](#footnote-2) as well as other well-known artificial datasets in research such as MADELON [[2]](#footnote-3). The program written in MATLAB[[3]](#footnote-4) generates an *M×N* dataset specified by the *num\_pat* and *num\_indep\_feat* parameters. The value in each feature is randomly drawn from a normal distribution. Out of these features, *n* of them (where *n* ≤ *N*) are standardised and the label is generated row-wise from them. This procedure tightly couples the label with the useful features. The variable *num\_useful\_feat* specifies the value of *n* in the program. The rest of *N-n* features represent the useless features.

In this study, 3 datasets were generated, each with 100 features with the following 3 ratios of useful to useless features: 70:30, 50:50 and 30:70. Each dataset has 500,000 datapoints with an approximate size of 350MB. The datasets were shuffled by rows and columns to ensure there are no hidden patterns in the data. This last procedure removed the bias that was initially seen in tests probably caused by the dataset generation program.

## Performance Metrics

As artificial datasets had been created, the useful and useless features were known prior to any feature selection. With this information, the features chosen by the different models (RF and chi-squared) could be compared against the known useful features and the feature selection accuracy calculated. This was achieved by taking the top *x* features from the feature importance rankings provided by both models, where *x* is the number of known useful features, and calculating the percentage of these that were the known useful features. This metric was also used for parameter tuning, by measuring how it changed as the parameters were changed.

In order to get a wider understanding of the impact of feature selection on the learning process, a linear SVC was trained using the features selected by the RF and the chi-squared test as well as on the dataset before any pre-processing, and a dataset of just the *known* useful features (i.e., with no data noise present). All parameters for the SVC were kept the same for each of the models.

It is important to note that the experiments were performed on the Databricks platform[[4]](#footnote-5), and thus time experiments were heavily dependent on the current workload on the cluster. Thus, time results are only to be considered relative to each other.

## Testing

### Parameter Tuning

To ensure that the filter and embedded methods were optimised to our datasets and for the purpose of this study, a selection of 2 tests was performed – aiming to find the best hyperparameter for each method through trial-and-error, where the values tested in each case were based on findings from previous studies. The best dataset – with 70 useful and 30 useless features – was used for both tests to reduce the risk of data noise affecting the results.

Test 1 was done for the filter method and served the purpose of finding the best number of buckets *numBuckets* for the discretisation of the datasets. It compares the feature selection accuracy and time with the following values of *numBuckets*: 100, 1,000, and 10,000, visualised in Tab. II.

Test 2 was done for the embedded method and served the purpose of finding the best number of trees for the RF implementation. On the one hand, a low number of trees risks losing out on information provided by some data points, as it is not guaranteed that they are selected by a tree during subsampling [36]. On the other hand, too many trees might not increase the performance but take longer to compute, which is especially critical when handling big data. Test 2 compares the feature selection accuracy and time with the following numbers of trees: 10, 20 (default), 30, and 100, based on what previous research indicated was reasonable, visualised in Tab. III.

### Effects of Data Noise

Test 3 was performed to discover how the feature selection accuracy of the two methods compared as the ratio of useful to useless features reduced, i.e., as more noise was introduced to the data. Firstly, each method was configured to its optimal parameter value for *numBuckets* and number of trees, respectively. Secondly, the feature ranking was calculated using the filter method and the embedded method. Thirdly, the top *x* features were selected from each dataset, where x is equal to the number of useful features present in the dataset. Lastly, the new subset from each dataset was split into training and test sets, according to 70% training data and 30% test data, and then used to train an SVC classifier to also see how the prediction accuracy changed as more noise was introduced. The computing time was also measured to help determine the scalability of the methods. See Tab. IV for test results.

### Baselines and Benchmarks

Additionally, in Test 4, the three datasets (each with a different ratio of useful to useless features) were used to model SVCs *without* any feature selection, providing a baseline to compare how much the pre-processing affected the results. See Tab. V for results from this test.

Likewise, in Test 5, the three datasets were also used to model SVCs with *perfect* feature selection, where the *previously* *known* useful features were selected (no feature ranking required as there are no useless features and therefore no data noise), providing a benchmark to compare how optimal each feature selection method is. See Tab. VI for results from this test.

# Results

The results from Test 1 (as shown in Tab. I) indicate that the computational time increases significantly – from 3.83 min to 15.19 min – with the number of buckets, in alignment with the findings from the research, despite *no* valuable changes to the feature selection (FS) accuracy. In fact, the lowest tested value of *numBuckets* (100) gave the highest feature selection accuracy of 68.57% - a relatively high rate. Therefore, it was deduced that 100 was sufficient for *numBuckets* in terms of discretisation of a big dataset, to maintain the scalability, i.e., speed, of the code.

# Table I: Chi-squared FS Accuracy and Time with Number of Buckets

|  |  |  |  |
| --- | --- | --- | --- |
| **No. of Buckets** | 100 | 1,000 | 10,000 |
| **Feature Selection Accuracy (%)** | 68.57 | 64.29 | 65.71 |
| **Time (min)** | 3.83 | 4.92 | 15.19 |

Differently, the results from Test 2 (as shown in Tab. II) indicate that the computational time is *not* significantly affected by the changes to the parameter, number of trees – the required computational power. This is because the time only increases by 0.59 min (~2-fold increase) with a 10-fold increase of the parameter. Therefore, this method is thought to be more scalable than the filter method when the data is continuous, and discretisation is required. Furthermore, the results are also an indication of a robust and stable solution, since the performance (i.e., feature selection accuracy) is *not* volatile with respect to hyperparameter tuning – with the feature selection accuracy remaining at a high rate of ~70% in all cases. These results are not in alignment with previous findings, which indicated that the RF parameters would be more strongly correlated with its performance. Evidently, the feature selection accuracy is optimal at 71.43% with a forest size of 30 trees.

To conclude the parameter tests, the final value of 100 buckets was chosen for the discretisation for the chi-squared test, and a size of 30 trees was chosen for the RF.

# Table II: Random Forest FS Accuracy and Time with Number of Trees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No. of Trees** | 10 | 20 | 30 | 100 |
| **Feature Selection Accuracy (%)** | 68.57 | 67.14 | 71.43 | 65.71 |
| **Time (min)** | 0.46 | 0.79 | 0.74 | 1.05 |

The results from Test 3 (as shown in Tab. III (a)) indicate that the chi-squared test’s feature selection accuracy decreases significantly with the prediction accuracy as more noise is introduced to the data, in alignment with previous findings [37]. As the ratio of useful to useless features decreases from ~2.3 (70/30) to ~0.4 (30/70), the feature selection accuracy decreases by 45.24%. Meanwhile, the prediction accuracy only decreases by 14.21%, indicating that the changes to the feature selection accuracy do not directly transfer to the final prediction accuracy. In fact, the first two test-cases show that a ~30% decrease in the feature selection accuracy only causes a ~10% decrease in the prediction accuracy. Moreover, the last two test-cases show that a ~15% decrease in the feature selection accuracy only causes a ~3% decrease in the prediction accuracy. This comparison brings the following two conclusions to light: the negative effect of data noise on both performance metrics affected the prediction accuracy by only ~1/3 of how much it affected the feature selection accuracy, and the more noise was added, the smaller the magnitude of change on the performance metrics. Generally, the performance of chi-squared test’s feature selection seems to be very volatile with respect to data noise, which is a critical element and common factor in big data problems.

With respect to how the data noise affects the RF feature selection in Test 3, Tab. III (b) shows that the feature selection accuracy is only marginally higher than the chi-squared rate for each dataset, indicating that both feature selection methods respond equally to the introduction of data noise. This was in alignment with previous findings about data noise in RFs, which claim that the noise transfers to all trees in the training process and the effects on the final RF are thereby maximised [38]. Regardless, previous findings also indicated that the RF should be expected to perform better than the chi-squared test. However, in this study, the RF seemingly followed the same trend as is illustrated by the chi-squared test with only a marginal improvement on the performance. Since Tab. III shows that the chi-squared test requires less computational time, it was therefore deduced that it is more suitable for big data problems. Nevertheless, RF is still the best-performing feature selection method with respect to feature selection accuracy, and in the case of a dataset with continuous values which need to be discretised, RFs also perform faster (as mentioned earlier).

# Table III: FS, SVC Prediction Accuracy and Time with Different Numbers of Useful and Useless Features. (a) Filter Method Top. (b) Embedded Method Bottom.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | Chi-squared | | |
| **Useful Features** | 70 | 50 | 30 |
| **Useless Features** | 30 | 50 | 70 |
| **Feature Selection Accuracy (%)** | 68.57 | 38.00 | 23.33 |
| **Prediction Accuracy (%)** | 80.33 | 69.49 | 66.12 |
| **Time (min)** | 0.09 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | Random Forest | | |
| **Useful Features** | 70 | 50 | 30 |
| **Useless Features** | 30 | 50 | 70 |
| **Feature Selection Accuracy (%)** | 70.00 | 40.00 | 26.67 |
| **Prediction Accuracy (%)** | 81.91 | 70.04 | 68.89 |
| **Time (min)** | 0.37 | | |

The results from Test 4, shown in Tab. IV, were namely the most unexpected set of results from the study. Firstly, the baseline tests show that changes in the prediction accuracy are seemingly insignificant, regardless of how much noise a dataset contains, contradicting Test 3 (Tab. III). Secondly, the prediction accuracy is significantly higher *without* any feature selection – i.e., with all useful *and* useless features present in the dataset during the training of the SVCs – compared to feature selection with RF and the chi-squared test. A reason for these results was thought to be that the feature selection methods missed selecting some very significant features – resulting in an impactful loss of information in the training stage of the SVCs.

As all the SVCs were trained on the same amount of data, the time did not change between the cases.

# Table IV: Baseline SVC Prediction Accuracy and Time without Pre-processing (FS) with Different Numbers of Useful and Useless Features

|  |  |  |  |
| --- | --- | --- | --- |
| **Useful Features** | 70 | 50 | 30 |
| **Useless Features** | 30 | 50 | 70 |
| **Prediction Accuracy (%)** | 99.52 | 99.51 | 99.56 |
| **Time (min)** | 34.48 | | |

The results from Test 5, as shown in Tab. V, were in alignment with the expectations based on the artificial dataset generation, and helped explain the results in Tab. IV. The results show that the benchmark prediction accuracy of the SVCs with only *known useful* features is almost 100% with all datasets. These results were thought to be caused by the following two reasons:

1. The lack of noise
2. The presence of all known useful features

Since Test 4 showed that noise had no significant effect on the performance of the SVCs, the second reason is believed to be the main cause of the successful results in Tab. V.

The labels in the artificial datasets are generated row-wise from the useful features and it is believed that training the SVM with all the useful features, regardless of the useful to useless feature ratio, significantly improves the prediction accuracy, explaining the extremely high accuracy rates of the results in Tab. IV. If at least one feature is not selected out of those useful features, the SVM will perform significantly worse as was seen in Tab. III. To test this hypothesis, the SVM was retrained with only 69 out of 70 useful features and the accuracy dropped to 94% - a 5% difference. When trained with even fewer useful features, the accuracy further decreased to 93%. These results confirmed the initial theory that the presence of useful features affect the performance of the final classifier more than the lack of useless features and that the low prediction rate of the SVCs with feature selection are caused by the chi-squared test and RF model missing useful features.

# Table V: Benchmark SVC Prediction Accuracy and Time without Useless Features

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Useful Features** | 70 | 50 | 30 | |
| **Useless Features** | 0 | 0 | 0 | |
| **Prediction Accuracy (%)** | 99.59 | 99.67 | 99.79 | |
| **Time (min)** | 32.91 | | |

# Conclusion

This contribution aimed to further explore embedded methods as a means of feature selection in big data, when compared to the current standard of filter methods. The results indicate that RF, the chosen embedded method, provides only a marginally better feature selection accuracy than the filter method chi-squared test it was compared to, and a significantly increased computational time. Furthermore, both methods appeared equally affected by the introduction of more noise into the dataset.

Although this may seem to support the favouring of filter methods, it is important to note the time taken to discretise the data. If included in the training time for the chi-squared test, it should be noted that RF then has a much better computational time. Moreover, increasing the number of trees in the RF did not have a significant impact on the training time, seemingly indicating that this approach would scale well as the dataset gets very large and more trees are required to get a good performance. As discussed in the study, RF are naturally suited to parallelisation, and the divide-and-conquer approach adopted demonstrates their ability to scale very well to large datasets. As a result, the conclusion is that embedded methods can be a viable alternative to filter methods, especially when dealing with continuous data or very large datasets and it is encouraged that more research be done into them for these purposes.

Regardless of the method, the greatest benefit of feature selection is seen in the reduction in training time between a dataset with no pre-processing, and one where feature selection has been performed, supporting the importance of this method to deal with some issues of big data.

Though the results don’t generally appear encouraging regarding the use of embedded methods, there are still further experiments that can be conducted to gain a better insight. The use of a real dataset could be explored to see if the results change once other issues, such as imbalanced and missing data, are potentially introduced. It may also be a point of interest to explore other embedded methods and ascertain whether they can provide a good performance if implemented with a divide-and-conquer approach.

# References

|  |  |
| --- | --- |
| [1] | G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Computers & Electrical Engineering,* vol. 40, no. 1, pp. 17-20, 2014. |
| [2] | R. Krishnan, V. A. Samaranayake and S. Jagannathan, “A Hierarchical Dimension Reduction Approach for Big Data with Application to Fault Diagnostics,” *Big Data Research,* vol. 18, p. 100121, 2019. |
| [3] | N. Kouiroukidis and G. Evangelidis, “The Effects of Dimensionality Curse in High Dimensional kNN Search,” *2011 15th Panhellenic Conference on Informatics,* pp. 41-45, 2011. |
| [4] | M. Rostami, K. Berahmand, E. Nasiri and S. Forouzandeh, “Review of swarm intelligence-based feature selection methods,” *Engineering Applications of Artificial Intelligence,* vol. 100, p. 104210, 2021. |
| [5] | N. K. Suchetha, A. Nikhil and P. Hrudya, “Comparing the Wrapper Feature Selection Evaluators on Twitter Sentiment Classification,” *2019 International Conference on Computational Intelligence in Data Science (ICCIDS),* pp. 1-6, 2019. |
| [6] | F. Haider, S. Pollak, P. Albert and S. Luz, “Extracting Audio-Visual Features for Emotion Recognition Through Active Feature Selection,” *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP),* pp. 1-5, 2019. |
| [7] | M. B. Çatalkaya, O. Kalıpsız, M. S. Aktaş and U. O. Turgut, “Data Feature Selection Methods on Distributed Big Data Processing Platforms,” *2018 3rd International Conference on Computer Science and Engineering (UBMK),* pp. 133-138, 2018. |
| [8] | G. Taşkın, H. Kaya and L. Bruzzone, “Feature Selection Based on High Dimensional Model Representation for Hyperspectral Images,” *IEEE Transactions on Image Processing,* vol. 26, no. 6, pp. 2918-2928, 2017. |
| [9] | H. F. Zhou, J. W. Zhang, Y. Q. Zhou, X. J. Guo and Y. M. Ma, “A feature selection algorithm of decision tree based on feature weight,” *Expert Systems with Applications,* vol. 164, p. 113842, 2021. |
| [10] | K. R. Pushpalatha and A. G. Karegowda, “CFS Based Feature Subset Selection for Enhancing Classification of Similar Looking Food Grains- A Filter Approach,” *2017 2nd International Conference On Emerging Computation and Information Technologies (ICECIT),* pp. 1-6, 2017. |
| [11] | S. Visalakshi and V. Radha, “A literature review of feature selection techniques and applications: Review of feature selection in data mining,” *2014 IEEE International Conference on Computational Intelligence and Computing Research,* pp. 1-6, 2014. |
| [12] | D. López, S. Ramírez-Gallego, S. García, N. Xiong and F. Herrera, “BELIEF: A distance-based redundancy-proof feature selection method for Big Data,” *Information Sciences,* vol. 558, pp. 124-139, 2021. |
| [13] | V. Vasudevan and D. F. Vinod, “A filter based feature set selection approach for big data classification of patient records,” *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT),* pp. 3684-3687, 2016. |
| [14] | S. Din, A. Paul, N. Guizani, S. H. Ahmed, M. Khan and M. M. Rathore, “Features Selection Model for Internet of E-Health Things Using Big Data,” *GLOBECOM 2017 - 2017 IEEE Global Communications Conference,* pp. 1-7, 2017. |
| [15] | V. F. Rodriguez-Galiano, J. A. Luque-Espinar, M. Chica-Olmo and M. P. Mendes, “Feature selection approaches for predictive modelling of groundwater nitrate pollution: An evaluation of filters, embedded and wrapper methods,” *Science of The Total Environment,* vol. 624, pp. 661-672, 2018. |
| [16] | Y. Fu, X. Liu, S. Sarkar and T. Wu, “Gaussian mixture model with feature selection: An embedded approach,” *Computers & Industrial Engineering,* vol. 152, p. 107000, 2021. |
| [17] | M. Lu, “Embedded feature selection accounting for unknown data heterogeneity,” *Expert Systems with Applications,* vol. 119, pp. 350-361, 2019. |
| [18] | T. Hamed, R. Dara and S. C. Kremer, “An Accurate, Fast Embedded Feature Selection for SVMs,” *2014 13th International Conference on Machine Learning and Applications,* pp. 135-140, 2014. |
| [19] | Y. Hua, “An Efficient Traffic Classification Scheme Using Embedded Feature Selection and LightGBM,” *2020 Information Communication Technologies Conference (ICTC),* pp. 125-130, 2020. |
| [20] | R. Genuer, J.-M. Poggi, C. Tuleau-Malot and N. Villa-Vialaneix, “Random Forests for Big Data,” *Big Data Research,* vol. 9, p. Big Data Research, 2017. |
| [21] | P. Płoński, “The 3 Ways To Compute Feature Importance in the Random Forest,” 2020 June 29. [Online]. Available: https://towardsdatascience.com/the-3-ways-to-compute-feature-importance-in-the-random-forest-96c86b49e6d4. [Accessed 23 April 2020]. |
| [22] | A. Altman, L. Toloși, O. Sander and T. Lengauer, “Permutation importance: A corrected feature importance measure,” *Bioinformatics,* vol. 26, no. 10, pp. 1340-1347, 2010. |
| [23] | N. Hoque, M. Singh and D. K. Bhattacharyya, “EFS-MI: an ensemble feature selection method for classification,” *Complex & Intelligent Systems,* vol. 4, no. 2, pp. 105-118, 2017. |
| [24] | L. Mitchell, T. M. Sloan, M. Mewissen, P. Ghazal, T. Forster, M. Piotrowski and A. Trew, “Parallel classification and feature selection in microarray data using SPRINT,” *Concurrency and Computation: Practice and Experience,* vol. 26, no. 4, pp. 854-865. |
| [25] | L. Raileanu and K. Stoffel, “Theoretical comparison between the Gini Index and Information Gain criteria,” *Annals of Mathematics and Artificial Intelligence,* vol. 41, no. 1, pp. 77-93, 2004. |
| [26] | T. Hastie, R. Tibshirani and J. Friedman, “Random Forests,” in *The Elements of Statistical Learning*, New York, Springer, 2008, pp. 587-604. |
| [27] | B. H. Menze, B. M. Kelm, R. Masuch, U. Himmelreich, P. Bachert, W. Petrich and F. A. Hamprecht, “A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data,” *BMC Bioinformatics,* vol. 10, 2009. |
| [28] | N. Rachburee and W. Punlumjeak, “A comparison of feature selection approach between greedy, IG-ratio, Chi-square, and mRMR in educational mining,” *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE),* pp. 420-424, 2015. |
| [29] | X. Jin, A. Xu, R. Bie and P. Guo, “Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles,” *Lecture Notes in Computer Science,* vol. 3916, pp. 106-115, 2006. |
| [30] | Y. Zhai, W. Song, X. Liu, L. Liu and X. Zhao, “A Chi-Square Statistics Based Feature Selection Method in Text Classification,” *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS),* pp. 160-163, 2018. |
| [31] | S. Ray, K. Alshouiliy, A. Roy, A. AlGhamdi and D. P. Agrawal, “Chi-Squared Based Feature Selection for Stroke Prediction using AzureML,” *2020 Intermountain Engineering, Technology and Computing (IETC),* pp. 1-6, 2020. |
| [32] | Apache Spark, “QuantileDiscretizer,” [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.QuantileDiscretizer.html. [Accessed 18 May 2021]. |
| [33] | “Apache Spark,” [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.QuantileDiscretizer.html. [Accessed 17 May 2021]. |
| [34] | Apache Spark, “ChiSquareTest,” [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.stat.ChiSquareTest.html. [Accessed 18 May 2021]. |
| [35] | D. Oreski, S. Oreski and B. Klicek, “Effects of dataset characteristics on the performance of feature selection techniques,” *Applied Soft Computing,* vol. 52, pp. 109-119, 2017. |
| [36] | L. Breiman, A. Cutler, A. Liaw and M. Wiener, “Package 'randomForest': Breiman and Cutler's Random Forests for Classification and Regression,” 25 March 2018. [Online]. Available: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf. [Accessed May 2021]. |
| [37] | R. Andrae, T. Schulze-Hartung and P. Melchior, “Dos and don'ts of reduced chi-squared,” 2010. |
| [38] | H. B. Li, W. Wang, H. W. Ding and J. Dong, “Trees Weighting Random Forest Method for Classifying High-Dimensional Noisy Data,” *2010 IEEE 7th International Conference on E-Business Engineering,* pp. 160-163, 2010. |

1. NIPS 2003 Feature Selection Challenge [↑](#footnote-ref-2)
2. <https://archive.ics.uci.edu/ml/datasets/madelon> [↑](#footnote-ref-3)
3. <https://www.mathworks.com/products/matlab.html> [↑](#footnote-ref-4)
4. <https://databricks.com> [↑](#footnote-ref-5)