Loughborough University

Analysing Networking Data to Detect Malware Attacks

By Chidera Uchenwoke

Student ID: B928510

22COP328: Data Science Project

Supervisor: Dr Yanning Yang

Submitted on 7^{th} September 2023

Abstract

There is an increasing necessity for the development of malware detection methods as malware continue to evolve rapidly. In particular, such methods which are effective on machines with limited computational resources available. The dimensionality of networking data used to develop malware detection tools as a result of the increasing sophistication of malware have also increased. Consequently this renders some traditional data pre-processing such as one-hot encoding, futile as it could lead to an explosion in dimensionality thus, severely diminish performance. Therefore, the aim of this project was to develop a malware detection method, using the 2019 Microsoft Malware detection dataset, which would require few computational resources without grossly diminishing classification accuracy. Hence, we developed and compared two filter-based input feature selection methods Joint Mutual Information Maximisation (JMIM) and Pearson's Chi-square with the LightGBM machine learning model to assess if either model was capable of achieving this standard. The results show JMIM with LightGBM significantly outperformed Chi-square with LightGBM in terms of classification accuracy. It also showed JMIM with LightGBM, on average, consumed 29% less memory and ran 5.3 times faster than Chi-square with LightGBM.

The notebooks submitted have a lot of Python packages dependencies. To avoid any issues, link(s) to view and run the project code online on the Kaggle notebooks have been provided:

- Control LightGBM: https://www.kaggle.com/code/deera25555/control-lgbm/notebook
- JMIM with LightGBM: https://www.kaggle.com/code/deera25555/jmim-lgbm/notebook
- Chi-square with LightGBM: https://www.kaggle.com/code/deera25555/chi-square-lgbm/notebook
- Wilcoxon test, data visualisation of training and validation results and test JMIM: https://www.kaggle.com/code/deera25555/test-jmim/notebook

Acknowledgements

I would like to express my sincere gratitude to Dr Yanning Yang. Her guidance throughout the project was invaluable.

Contents

1 Introduction

1.1 Motivations

1.1.1 A brief evolution of malware detection methods

Malware is software developed with intent to; disrupt network transmission, usurp machines to make use of its resources and or extract sensitive data without the knowledge or consent of the user [1]. Earlier implementations were detected by identifying signatures, a sequence of instructions written in assembly, that were known to execute said malicious acts [2]. In recent years, however, malware developers have invented more sophisticated obfuscation techniques to evade this method of detection. The inception of polymorphic malware, for example Ghost/Win32, saw malware that could generate millions of unseen malignant mutations under the guise of solely completing the task the user had originally intended the software for [3].

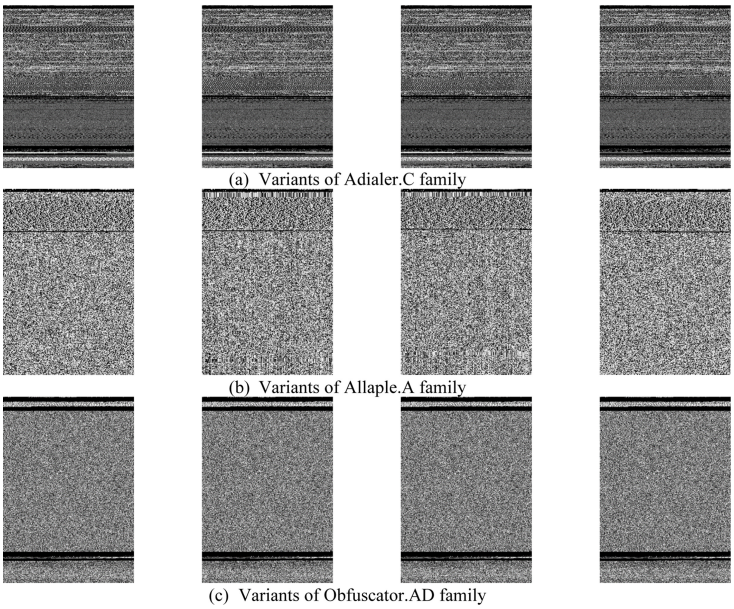

Figure 1: Example of a polymorphic malware family variants transformed into images by a custom neural networks to highlight their structural similarities [4].

Malware detection methods then began to rely on observing the behaviour of the executing code as well as its signatures. Unfortunately, some behaviours of polymorphic malware were only triggered under very specific system environment conditions which were difficult to reproduce for testing [5]. There have since been several attempts to mitigate this using statistical learning ('machine learning') models i.e., Support Vector Machine (SVM), Linear and Logistic Regression [6] [7] [8]. In addition, at the time of this report, the AV-TEST Institute registers approximately 2.9 new malware samples per second, each with their own sophisticated mechanisms to avoid detection [9]. Instead of having to keep at pace with the extensive, changing domain knowledge, machine learning models infer patterns in a sample through trainable parameters thus enabling the

detection of unseen variants [10]. This, however, introduces a new challenge – current statistical learning models are time consuming and expensive. Often, they require; an external graphic processing unit (GPU), specific proprietary software and or specialists who know how to use and maintain them [11] [12] [13].

1.1.2 Input feature selection: A necessary prerequisite for statistical learning models?

One method to amend the aforementioned is input feature selection. By specifying the explanatory features relevant to the class label, it allows the problem to be interpretable. That is, the ability for a person to explain or represent the model's 'decisions' in understandable terms so as to identify and remedy model biases [14]. Input feature selection is that of finding a subset such that the statistical learning model run on it returns the highest possible accuracy [15]. (Note that it does not construct new features.) As a result, demonstrated by [16], [17] and [18] it can substantially reduce the overall execution time, memory, bandwidth and other resources utilised to facilitate detection. [19] also found that it reduced the error rate of the C4.5 classifier from 24.3% to 11.1% if out of the original 6, 3 relevant features of the MONK-1 dataset [20] were provided instead. Input feature selection is also the solution to a limitation of The Laws of Large Numbers (LLN) theorem. LLN underpins a broad class of statistical models such as the Monte Carlo method [21]. The theorem demonstrates if $\mathbf{n}$, the number of trials, is sufficiently large, where $\mathbf{n} \to \infty$, the results of the model converge and become stable or 'predictable' in the long-term [22]. The limitation is that if the data follows a heavy-tailed distribution, as is typical for high dimensional data to do so, the results of the model it is used for may fail to converge [23]. This poses a problem because as $\mathbf{n}$ increases the better the approximation is supposed to be, and reduce the influence of model bias. High dimensional data would not see the reduction of biases even as $\mathbf{n}$ increases unless the number of features have been reduced often via input feature selection.

Bio-informatics is one of many fields where extreme cases of high dimensional datasets are frequent as, alike network telemetry data, the number of experimental units is small relative to the underlying dimension. An illustrative example is the human genome. One property of the human genome is Single Nucleotide Polymorphism (SNP), the change of a single DNA sequence building block, and it accounts for the majority of genetic variation between humans. An example of an SNP is as follows - suppose you have a DNA sequence, AACGAT. The substitution of G for C that produces AACCAT is an SNP. (If this is of interest, a more rigorous explanation and examples of SNP can be found in [24]). There are 10^6 unique SNP's on average present in a single human genome [25] [26]. The human genome data types and the extent of data acquisition and computing facilities has only increased over the years [27]. Thus, further reinforces input feature selection is obligatory to develop reliable, statistical models for current and future data problems.

1.1.3 Preliminary design considerations

In the design of the experiment, a challenge we consider is that the hardware resources of machine used to run the experiments on is limited given the size of the dataset. [1], who conducted a similar experiment did not have this limitation and other papers such as [28] did not disclose the resources they used to experiment with this dataset. Therefore it was difficult to gauge what would be required prior experimentation and whether we had the resources available to do so. To overcome this we considered generating a similar, manageable synthetic dataset as [29] did. Synthetic data can reduce the time for model development as the data would not require pre-processing. The downside it is difficult to accurately mimic outliers. It is recommended thus, as by [30], to test the model on real data prior release if synthetic data was used for testing. Another alternative would have been to take a smaller sample of the dataset, however, this would result in a less generalisable model. The dataset adopted for this study is only applicable to Windows machines which, as consequence, further reduces the scope of application. Joint Mutual Information Maximisation (JMIM) one of the input selection methods, has been used for multiple intrusion detection studies, however, intrusion detection and malware detection are not synonymous. Malware is a type of intrusion which requires a host machine and not based on network traffic alone [31]. Hence, another challenge we face is that JMIM has not been implemented on the competition dataset before and very few network-based malware detection datasets at the time of writing. To tackle this we introduce a control experiment, without input feature selection, to benchmark JMIM and a second input feature selection method which has been used before against. Lastly, we assume all class labels are correct and have been 'exhaustively defined' [32], i.e, there is no ambiguity and all possible labels are present in the training stage of analysis.

In summary, malware detection datasets are increasingly large and high dimensional. This coupled with current machine learning models make finding solutions to modern data problems slow, computationally expensive and limits the number of people who can afford to test them. Hence, the aim of this project was to develop a lightweight, computationally inexpensive machine learning model for malware detection with the considerations outlined in Section 1.1.3 in mind. To achieve this:

- We implement two methods of input feature selection, one new and one used before, to reduce the size and dimensionality of the dataset.
- Then, we select a machine learning model with historically good results on the dataset to run the instances of the selected features on.
- After which we compare the two methods of input selection in terms of; the number of features selected, execution time, memory consumption and classification accuracy.

The rest of this paper is organised as follows. In Section 2 we define the task of selecting optimal features, review and compare filter and wrapper-based approaches. In

Section 3 we describe the method constructed to develop the machine learning model and accompanying input feature selection method. In Section 4 we show the results of our experimentation. Lastly, in Section 5 and 6 we summarise and discuss the implications of our results and recommendations for future work.

2 Background

2.1 Wrapper methods

2.1.1 Generic design

Figure 2 shows the general schema of wrapper methods. Wrapper methods are a type of input feature selection comprised of four components; a machine learning model, a search algorithm, a sample of instances with class labels and a sample of test instances [19]. First, a machine learning model is run on a subset of instances of the features

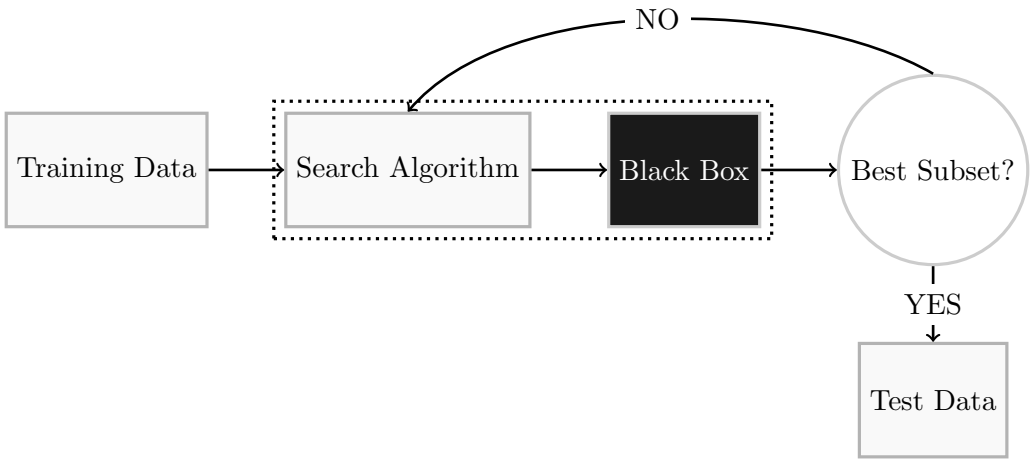

Figure 2: Generic framework of wrapper methods.

selected by a search algorithm that have been partitioned into a training and validation set. The purpose of the training set is to adjust the user defined parameters of the model using iterative optimisation methods such as Cauchy's gradient descent [33]. The purpose of the validation set to fine-tune the model and assert if it has overfitted. Over-fitting occurs when the model has adjusted its parameters such that it can no longer generalise to the overall population, rather, it is only applicable to the sample [34]. For example, if the model was developed to distinguish between cats and dogs and the training set only comprises of Bull dogs and Main Coon cats, the classification accuracy of the test set would decrease with different breeds of cats and dogs since they have not been seen before. After the machine learning model has run, the subset of input features with highest evaluation after repeated runs is evaluated on the test set. Test set is completely separate to the training and validation instances to provide an unbiased evaluation of the black box. It may also contain features which have never been seen before. Let us use [35] as a brief example: the black box is SVM, the search algorithm is Recursive Feature Elimination

(RFE), the 38 samples of training instances and class labels were gene expression values and two variants of leukemia respectively. Lastly, the test set contained 34 samples of the same type of data in the training and validation set.

2.1.2 Search algorithms

Input feature selection is inherently a combinatorial problem since the goal is to find an optimal subset. The search algorithm, therefore is important this search, for wrapper or filter methods (see Section 2.1.2) is as efficient as possible. We walk-through an example to compare a naive search with a search algorithm to show how much more efficient it is.

Suppose we have set with 3 features, $N = \{A, B, C\}$ and list all of its possible unique subsets:

$$\emptyset, \{A, B, C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A\}, \{B\}, \{C\}$$

One can see, the total number of unique subsets is $8 = 2^3$. Formally, let N be the number of elements in a set and K be the number of elements in a subset. The total number of unique subsets is $\binom{N}{K}$, where $0 \leq K \leq N$ (see Binomial Coefficient). For example if $K = 0$ i.e., when there are 0 elements in the subset, the total number of unique subsets is 1, an empty set ($\emptyset$), as $\binom{N}{0} = \frac{N!}{K!(N-K)!} = \frac{N!}{0!(N-0)!} = \frac{N!}{N!} = 1$. If we extend this example to up to the maximum of K, which is when the number elements in the subset K is the same as number of elements in N we have:

$$\underbrace{\binom{N}{0} + \binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{N}}_{\text{N subsets checked}}$$

Then, by Addition Principle:

$$\binom{N}{0} + \binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{N} = 2^N$$

Thus, the only apparent way is to search all possible subsets of N is to search 2^N possible subsets of N. Note, that it makes little sense for a search algorithm to evaluate an empty set of features so the total number of possible subsets to assess is $2^N - 1$. Now consider backward elimination, of which RFE is derived from. The search algorithm begins with all features then, at each iteration, excludes one feature. The feature excluded is the one which returns the worst result based on an evaluation criteria selected by the user. For instance, in [36], this was the predictive residual error of sum of squares (PRESS) - the overall difference between the true class labels and class labels predicted by a model. Backward elimination only needs to search set N, K times where we assume we cannot check the same feature more than once. Therefore, $K = N - 1$ and the number iterations required to find the optimal subset of features is, by Multiplication Principle, $N \cdot (N - 1)$. This is considerably faster than the naive approach and without search algorithms such as these, both feature selection methods would be computationally prohibitive when N is large.

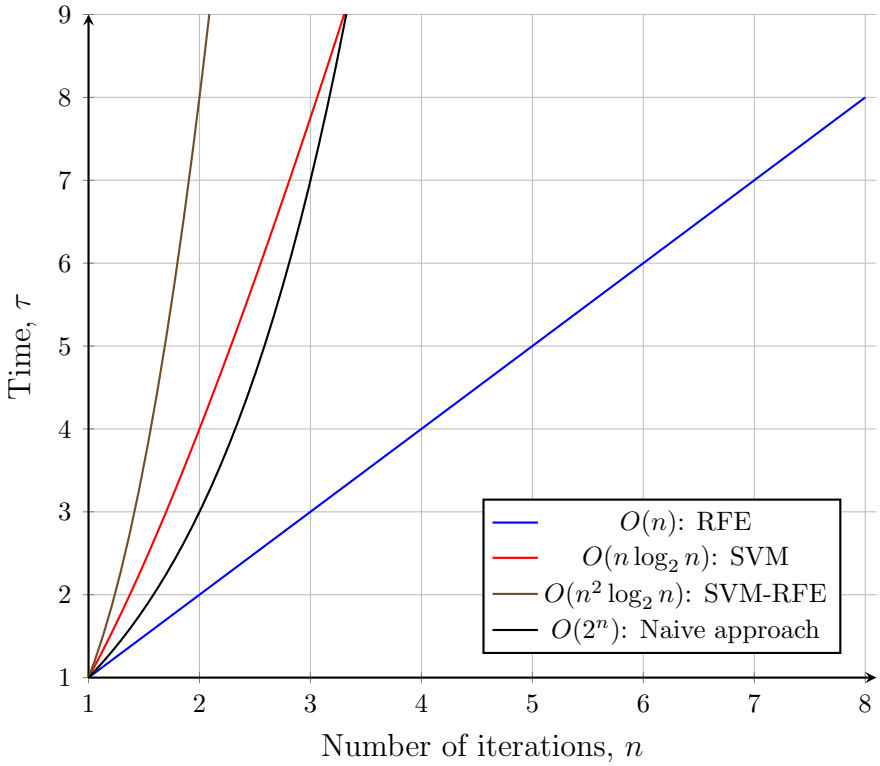

Figure 3: Time complexities of RFE, SVM, SVM-RFE and the naive approach.

2.1.3 The practicality of wrapper methods: SVM-RFE

In practice the additional complexity of the machine learning model, make wrapper methods time-consuming process compared to filter methods. For example, [37] demonstrates the time complexity of the SVM and RFE in the SVM-RFE wrapper which is the time it takes for the model to run as a function of the number of elements it must iterate over. SVM has a time complexity of $O(n \cdot log_2 n)$ and RFE has $O(n)$. Thus, the resulting combination of the two is $O(n^2 \cdot log_2 n)$.

This is expensive however, if there are sufficient resources available for the task, wrapper methods often return more accurate results [38]. Another issue with wrapper methods like SVM-RFE is the reliance on models for input feature selection which remain mostly black box. A black box is a mechanism with an input-output relation whereby meaningful explanations of its internal logic is absent [39]. This is of concern as we risk creating and using systems that we are unaware of the full extent of its vulnerabilities or are unsafe [40]. To date, there are only a few recognised interpretable machine learning models: decision trees, rules and linear models [41].

2.2 Filter methods

2.2.1 Generic design

Filter methods do as the name suggests - they filter out poorly informative features based on their statistical properties independent of the machine learning model adopted [42].

The generic design of filter methods, compared to wrappers is simpler as it only comprises of two components: a search algorithm and evaluation function. The evaluation function at least compares features the class label to deduce which set of features should be made redundant [38]. The search algorithm then optimizes this by searching for features based on the previous result rather than unnecessarily, as shown in Section 2.1.2, searching all possible contenders again.

2.3 Applications of filter and wrapper methods for malware and intrusion detection

The vast majority of literature still remain somewhat undecided which method is generally, 'better'. In Section 2.1.3 we mention this is due to the speed/memory to accuracy trade-off. Thus, it is up to the user to decide based on time and availability of resources which model best facilitates their desired outcome(s) [29]. This can challenging so current research is focused on exploring each subdomain. For example, [43] develop the Genetic Algorithm Logistic Regression (GA-LR) wrapper method for intrusion detection on the UNSW-NB15 and KDD99 dataset. In contrast, [44] proposed the Chi-square filter method to train an Logistic Regression model on the UNSW-NB15 dataset. Interestingly, the wrapper method is able to run on a machine with a less powerful CPU and with 4x less GB of physical RAM. This is a direct contrast to [38] and many others who found, in general, wrapper methods require more resources. They argue such an outcome was made possible by implementing the wrapper method in the C++ programming language instead of Python. This is backed by [45] who compared C++ and Python based on the average execution time and memory consumption of a binary search. They too found C++ was substantially more efficient while Python was simpler and intuitive therefore faster to implement. [43] also fast-tracked development by using pre-built modules to perform matrix transformations as opposed to programming the operations ab initio.

Although applied to the same dataset (UNSW-NB15) the only shared performance metric that allowed the evaluation of each study to be comparable was the binary classification accuracy. The Chi-square filter method coupled with Logistic Regression achieved a higher average binary classification accuracy of 98.17% with the selected features whereas the GA-LR wrapper achieved 81.42% on the test set. This could be because the filter method selected 3 more features than the GA-LR so the model had an additional $175,341 \times 3$ more instances to retrieve information from. Other than using the Python programming language, this could also contribute to amount of computational resources required in the end. Another reason may lie in the features each model selected as some may be more informative than others for the Logistic Regression. To derive an explanation without expert knowledge on intrusion detection, is a reminder of the inherent disadvantage of black boxes explained Section 2.1.3. The GA-LR wrapper method performs significantly better on the KDD99 dataset although according to [46] the dataset, KDD99, is outdated. [45] make light of this as old datasets on machine learning models 'limits their detection

accuracy'.

The Chi-square method [45] adopts is (1) robust with respect to the distribution of the dataset and (2) simple therefore computationally inexpensive to compute [47]. However, it is unsuitable for high cardinality features, that is, categorical features with 20 or more categories such as the 'CityIdentifier' feature in this project dataset and has the underlying assumption that all features are independent. Another downside is it can only be applied to discrete features therefore continuous features must either be transformed or discarded. For this research project, however filter methods are still favoured over wrapper methods because they do not require another classifier i.e., decision tree, and are more efficient thus less demanding to run [48]. Thus following [28] because, to our knowledge it is the only paper that has applied this method to this dataset, Chi-square was selected as the first filter method for experimentation. Alongside Chi-square, Joint Mutual Information Maximisation (JMIM) will be adopted based on the findings of [48] for comparison. They found their proposed JMIM was the best filter-based feature selection method for classification tasks with high dimensional datasets compared to traditional methods such as Mutual Information (MI). JMIM also removes the unrealistic assumption the features of real network telemetry data are linearly correlated [49]. Unlike other mutual information based filter methods designed to handle high dimensional data such as Fast Correlation-based selection, it considers the variation of one feature with respect to another known as the 'interaction' rather than solely the class label [29].

2.4 Machine learning methods for malware detection

For both filter and wrapper methods it is important to select which machine learning model may be best suited for the problem. This could be by the assessment of multiple models in a single study. The focus of this paper however, is to compare feature selection methods so a review of other papers to select the model that consistently produced 'good' results for malware and intrusion detection was ultimately the model selected for this project.

[1] assessed the performance of Light Gradient-Boosting Machine (LightGBM) with K-Fold Cross validation, LightGBM fitted to a Sparse Matrix, a Decision Tree classifier, and a neural network using the Microsoft Malware prediction dataset. Their results showed both LightGBM models outperformed the Decision Tree classifier and neural network in terms of AUC score and execution time. Notably, the LightGBM fitted to a Sparse Matrix slightly outperformed LightGBM with K-fold classification. In brief, a sparse matrix is a matrix where the majority of the matrix elements are zeros therefore by nature are more easily compressed and require less memory to store [50]. Dense matrices i.e., the Microsoft Prediction dataset can be encoded to form sparse matrices. However, there is no description of the hardware resources, programming language(s) or third-party software packages utilised. As consequence, their experimental results are difficult to reproduce. A simple work around would be to directly contact the researchers via the emails provided

within the paper to request this information.

[28] compared k-nearest neighbours (KNN), support vector machine (SVM) and LightGBM for malware detection on the same dataset. The Chi-square filter method was adopted to reduce the number of features by computing the association between features and discarding features with weak association. The main advantages of this method is detailed in Section 2.3. The study also does not take the occurrences of each feature into account. For example, a feature that has appeared infrequently could still be useful for malware detection. Despite this, [28] also demonstrated the best performing model was LightGBM also based on its AUC score and execution time, thus consistent with [1].

[51] also evaluated the performance of Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory recurrent neural networks (LSTMs) and LightGBM for malware detection using 6 datasets each consisting of 15,840 observations. Their aim was to classify and analyse anomalies present within the datasets across the 11 days 'Westrock', an American packaging company, were subject to malware attacks. Contrary to [1], they found in most cases neural networks such as RNNs and Bidirectional RNNs (Bi-RNNs) outperformed LightGBM. However, this may be attributed to the features used for detection. For example, while [1] found LightGBM detected malware based on the properties of the Windows machine such as its default browser and version of operating system, RNNs and Bi-RNNs detected malware based on the internet routing observations such as the number of announced and withdrawn IP prefixes over time. Unlike LightGBM, RNNs and Bi-RNNs are better suited to analyse time-dependent data because they are capable of identifying long-term dependencies to use for prediction [52]. In addition, the combined observations of datasets in [51] are considerably smaller than [1] and [28] which both comprise of 16,774,746 observations for training and testing. The difference may also lie in their partitioning of the training and testing data. With smaller datasets a proportion of training data larger than 60% may have enabled the model to 'learn' more about the features and produce more accurate classifications. Both [51] and [28] found that generally best performance for LightGBM was achieved when the dimensions were reduced.

2.5 Other applications of LightGBM

LightGBM has been used in a plethora of other fields to solve classification problems. [53] uses LightGBM to predict whether toxicology endpoints were 'safe' or 'toxic' based on mutagenicity datasets. In brief, mutagenicity are chemical compounds that induce genetic mutations. They found LightGBM was about 37 times faster than the Deep Neural Network (DNN) they had simultaneously conducted the experiment on. It also produced higher classification accuracy than the DNN as is consistent with [54] who used compared LightGBM, DNN (as well as Random Forest (RF), XGBoost and CatBoost) for predictive pharmaceutical models. They also find LightGBM achieved a higher accuracy and took significantly less time to execute. For example, they found LightGBM to be 1000

times faster than RF and 4 times faster than XGBoost. [55] go as far as to exclude DNN and SVM completely from experimentation because the process to search for optimal model parameters and time to train the final model would require a significant amount of time which was not available. Thus, LightGBM fits well with the aim of the study which is to develop a model for machines with limited resources.

3 Method

3.1 Environment Setup

3.1.1 Microsoft's Malware Prediction Dataset

The Microsoft Prediction dataset will be utilised. It was made available for a Kaggle competition in 2019. The network telemetry data was acquired using Microsoft's proprietary anti-malware software - the Windows Defender. The goal of the competition was to predict the probability of a Window machine being infected by malware based on its properties. Both datasets are large - there are 8,921,483 instances in the train dataset and 7,853,253 instances in the test dataset. Both datasets comprise of 82 unique features, not including the binary class label, 'HasDetections'.

Link to the dataset: https://www.kaggle.com/c/microsoft-malware-prediction/data.

3.1.2 Machine specifications

The specifications for the machine the experiments were conducted on is shown in Table 1 below:

Table 1: List of relevant machine specifications.

Property	***Quantity / Description***
Physical RAM (GB)	8
CPU	i7-8650U @ 1.90GHz
Number of CPU cores	4
Number of logical processors*	8

* The higher the number the more instructions (threads) can be processed simultaneously. Section 3.3 shows how this was utilised to reduce the execution time of the whole program.

3.1.3 Programming language & software modules

Although C++ is generally considered a more efficient programming language than Python, the latter (version 3.7) was selected for this project. As discussed in Section 2.3, the Python programming language is more intuitive thus it would take less time to implement a number of open-source machine learning models. It would also be easier for other users to implement and validate the results themselves. This is because all of the software modules used to conduct analysis including `numpy, gc, sklearn`, are widely used and have been thoroughly tested. `numpy` was used to implement linear algebra algorithms and operations optimally such as computing the dot product of two arrays. `gc` was used for garbage collection. This frees memory taken up by empty lists even though they appear to have no memory ramifications. These lists have not been deleted but are no longer of use for the rest of the program. We use `sklearn` to implement a number of machine learning tools such as `train_test_split` to partition the train dataset into train and validation subsets. Another reason is since the JMIM algorithm has not been implemented before on this dataset it was important in this project to ensure the user could follow the code with as much clarity as possible. Python syntax is simpler than C++ for example in Python initialising variable `num` as 9223372036854775808 is:

```
num = 9223372036854775808
```

Python has in-built functionality to deal with large numbers whereas C++ you must specify the correct size of integer for initialisation:

```
unsigned long long int num = 9223372036854775808;
```

A link to C++ fundamental data types: https://en.cppreference.com/w/cpp/language/types.

3.2 Exploratory Analysis

This analysis was conducted to gather information about the data types and structure of the dataset to determine which pre-processing methods would be necessary to implement before the machine learning model was run on the dataset.

3.2.1 Class distribution

The class label is binary, that is, the machine is able to detect malware, which was represented numerically with a 1, or not, which was represented numerically with a 0. It was important to check if the dataset was balanced, meaning there number of machines with a class label of true and false were equal. While this does not express a realistic distribution of host-based network telemetry data, where the number of benign instances are typically much larger than the malign instances, it does reduce bias during training [56].

3.2.2 Proportion of data types

The proportion of data types was computed to determine which filter-based methods could be applied to the dataset. A dataset with majority numerical float data types could have linear and non-linear methods such as Principle Component Analysis (PCA). PCA however, it not suitable for binary and or features that contain only non-negative integer values due its underlying assumption that the instances of the data type follow a Gaussian distribution [57]. To accommodate these types it would have to assume a Bernoulli or Poisson distribution for binary and non-negative integer datasets respectively.

3.2.3 Proportion of missing values

The proportion of missing values were computed and visualised to identify which features had more than 95% of its values missing. It was important to not remove features before identification of high cardinality features to avoid the accidental removal of a potentially informative feature despite having a large proportion of missing values. Step 4 of Section 3.3 explains this further coupled with an example.

3.2.4 Cardinality

High cardinality was a major limitation in the study of [28]. Once the categorical features were identified and separated in Section 3.2.2 the number of distinct categories in each feature were counted however, only the features we were interested in, the number of features of which satisfied the criteria, were returned. As mentioned in Section 2.3 we were only interested in features of high cardinality therefore features with more than 20 distinct categories.

3.2.5 Linear correlation of numerical features

Pearson's coefficient of correlation, r (see Equation 1) was used to assess the linear association between two numerical features. Values range from -1 to 1 inclusive, where higher values indicate better agreement [58].

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1}$$

Where:

- x_i and y_i are the individual instances of each feature x and y.
- n is the total number of instances in a feature. The total number of instances of feature x must be the same as the total number of instances of feature y.
- $\bar{x}$ and $\bar{y}$ are the mean of features x and y respectively.

The aim of this assessment was to identify the presence of collinearity within the group of numerical features. Two numerical features are said to be collinear when they are are

strongly correlated, that is, the Pearson's coefficient of correlation is greater than 0.7 [59]. This means it is likely if one feature is deemed important the inclusion of the other would not add to or improve the model. The addition of both collinear features in model may also reduce the statistical significance of one which should be significant [60]. Note while this method is a simple to compute and plot, it assumes features are homoscedastic. A homoscedastic distribution would show the residuals, e_{xy} are the same distance from the mean of residuals $\bar{e}_{xy} = 0$ [61]. Pearson's correlation coefficient also assumes features are independent and only assess the linear relationship between features which may instead have a non-linear relationship.

3.3 Pre-processing

The dataset must be pre-processed before the model or input features selection methods are run on it for best results. Pre-processing involves data cleaning which aims to remove or transform features and or instances which would undermine the performance of the model. For this study it also involves introducing ways to reduce the amount of memory allocated and execution time. For example, some functions were executed by running parallel for loops using the `Parallel` method in the `joblib` Python package. By distributing the looped function across all available CPU cores instead of one it faster than it would be to the execute the same function linearly. A caveat discovered of `Parallel` is that it requires consumes more memory than the linear approach. As consequence, in some cases, it led to a memory leak which forced the program to terminate. Thus, it was only applied if the function itself did not require large amounts of memory. Below is a line from the program to show how the `Parallel` method was used to discretize continuous features prior to JMIM for reasons detailed in Section 3.4.1:

```
new_df = Parallel(n_jobs=2)(
        delayed(equalFrequencyDiscretisation)(f, df, num_bins)
        for f in features
    )
```

Where:

- `n_jobs` is the number of CPU cores for parallel processing to be conducted on.
- `f, df, num_bins` is the feature, dataset and number of bins to discretize the feature instances into respectively.
- `new_df` is the fully discretized dataset returned once all the operations of the function have been executed.

To clean the data, the methodology in [28] will be partially adopted because it uses the same dataset and proposes cleaning methods which remove noise and simultaneously reduce memory usage.

1. Cast each feature value into smaller, better fitted data types. For example, float64 uses 64 bytes of contiguous memory whereas float32 uses only 32 bytes.

2. Remove features where the NaN (missing) values were more than 95% and were not identified have high cardinality. The only feature which satisfied this criteria was 'Census_ProcessorClass'.

3. Label encode categorical features instead of one-hot encode as it would lead to dimension explosion [62]. Label encoding replaces categorical values with numbers from 1 to N-1, where N is the total number of values the feature can have.

4. Frequency encoding is a technique used to convert categorical feature values to numerical values based on how frequent each unique category occurs [63]. The idea is that the frequency of some categorical feature values may have an affect on the value of the target variable 'HasDetections'. For example, if a particular category of a feature appears rarely or has a high proportion of missing values but often results in 'HasDetections' = True it should be further investigated and not discarded prematurely during cleaning. Frequency encoding also reduces the effects of high cardinality features. Without a method to manage these features, to store and process every instance of category during training would expensive [64].

5. Duplicates were removed from the dataset as they can potentially over-emphasise the importance of features [65].

6. Instances comprised of entirely missing values were removed to prevent any unnecessary noise which is known to reduce model accuracy [66].

7. Five ordinal features were identified: 'EngineVersion', 'AppVersion', 'AvSigVersion', 'OsVer' and 'Census_OSVersion'. Each number represents the version number of the product. Reasonably, the assumption was the closer the version numbers are to each other the more similar the machines are likely to share similar vulnerabilities. Using 'EngineVersion' as an example, engine version 1.1.15100.1 shares more properties with 1.1.15200.1 than 1.1.14600.4 since the distance from 15100 to 15200 is shorter than 15100 to 14600. The release notes for later Windows Defender engine versions detail the types of malware each engine version is capable of detecting. For example, engine version 1.395.1289.0 can detect a number of severe malware such as Win32/CryptGen!MTB. The features were separated, using '.' as the deliminator, into separate columns because naive label encoding would assign an entire version to single number. By separating them the information between the distances is maintained.

Note that step 6 temporarily increased the dimensionality of the dataset so the final step of the pre-processing was to apply the cast function (step 1) to the dataset again. As a result of pre-processing from step 1 to 6, the memory of the original train dataset was reduced by 62.6% (from 3709.57 to 1386.83 Mb).

3.4 Selection of K input features

The size of the optimal subset of features is user-defined. Hence, a range of different K values for the subset size of the 'best' features for both input feature selection methods was utilised. This was also necessary to assess whether any differences within the results were significant or not the statistical test described in Section 4.6. To create the control experiment the next two subsections were omitted as the the aim of the control experiment was to see if results improved with feature selection compared to without.

3.4.1 Joint Mutual Information Maximisation

Joint Mutual Information Maximisation (JMIM) falls under the domain of mutual information which has been widely used by itself or as the foundation of other filter input selection methods. Mutual Information (MI) is an estimate of the measure of information shared or association between two random features [67]. Or in the case of this project, the amount of we known about the class label, 'HasDetections' given a random feature in the dataset. By definition, this phenomena appears similar Pearson's linear correlation mentioned in Section 3.2.5, yet it does not suffer any of the same limitations. The upper limit of MI is the minimum entropy of the either the feature or class label as shown in Equation 4 [68]. The lower limit is zero, where two random variables are said to be independent if and only if their MI is zero. Entropy is a measure of the average amount of information a feature conveys to represent an event where all possible outcomes have been considered [69]. Given the probability density function (PDF) is $p(x) = P\{X = x\}$ the entropy of feature $x \in X$ of size N_x is defined as:

$$H(X) = -\sum_{i=1}^{N} p(x) \log p(x), \quad H(X) \geq 0. \tag{2}$$

Joint entropy is the entropy of two random variables. In this project this is the feature and class label. The joint PDF is therefore $p(x, y) = P\{X = x \text{ and } Y = y\}$ and the equation becomes:

$$H(X, Y) = -\sum_{i=1}^{N_x} \sum_{j=1}^{N_y} p(x, y) \log p(x, y), \quad H(X, Y) \geq 0. \tag{3}$$

Customarily, we use log of base 2 [70], [71]. Low or minimal entropy indicates the feature has a homogeneous sample of instances. On the contrary, high entropy indicates the sample is diverse so has many distinct categories and thus there is greater uncertainty. For example, each face of fair dice has a $\frac{1}{6}$ probability of being selected. Therefore it is said to have a higher entropy than a fair coin where each face has $\frac{1}{2}$ probability of selection.

Now using Equation 2 and Equation 3, MI is mathematically as follows [72]:

$$I(X; Y) = H(X) + H(Y) - H(X, Y), \quad 0 \leq I(X; Y) \leq min(H(X), H(Y)) \tag{4}$$

Where $H(X)$ is the entropy of a given feature, $H(Y)$ is the entropy of the class label and $H(X, Y)$ is the joint entropy of the feature and class label. MI is also symmetric so

$I(X;Y) = I(Y;X)$.

We also checked the results of mutual information on the dataset against the results given by `mutual_info_score` function from the `sklearn` Python package. Since the package uses the natural logarithm instead of log base 2 our MI was temporarily adapted to the natural logarithm to see if the values were an exact match, which was the case. From this Mutual Information Maximisation (MIM) was introduced in 1994 by [73]. It uses the forward selection search algorithm and MI as the evaluation function (see Generic design of filter methods in Section 2.2.1) to return the maximum entropy. However, it was later found to be sub-optimal for high dimensional datasets thus unsuitable for this project [74].

Joint Mutual Information unlike MIM also considers the mutual information between the feature currently being assessed and the subset of previously features [75]. Only if the feature is complementary, that is, has the highest mutual information between the original features, class label and subset of selected ('most informative') features, it is then added to the subset of selected features. Thus to compute JMI it requires the computation of entropy of a given feature, the entropy of the feature and the class label (joint entropy) and the entropy of the feature, class label and the features in the subset of selected features (trivariate entropy) as shown in Equation 6. Given the trivariate PDF is $p(x, y, z) = P\{X = x \text{ and } Y = y \text{ and } Z = z\}$ trivariate entropy is defined as:

$$H(X,Y,Z) = -\sum_{i=1}^{N_x}\sum_{j=1}^{N_y}\sum_{k=1}^{N_z} p(x,y,z)\log p(x,y,z), \quad H(X,Y,Z) \geq 0. \tag{5}$$

In the extensive review of [76], JMI outperformed MIM as well as 7 other filter methods in terms of accuracy and stability on the GISETTE and MADELON datasets (datasets for handwriting tasks). Below is the equation for JMI:

$$\begin{aligned} &I(X,Z;Y) = H(Z) + H(X,Z) - H(X,Y,Z), \\ &0 \leq I(X,Z;Y) \leq min(H(X), H(Y), H(Z)). \end{aligned} \tag{6}$$

Joint Mutual Information Maximisation was later instantiated by [48]. The main difference between JMI and JMIM is the feature with the maximum JMI is selected at each iteration whereas JMIM selects the feature with the minimum JMI of the original set of features into set we called jmi. This set is essentially a set that contains the jmi of each feature. Since the minimum JMI is input at each iteration this results in set jmi sorted in ascending order with respect to the JMI values. The maximum index therefore contains the maximum JMI which is then returned. In brief, the JMIM of a given feature, $x \in X$ not already in the subset of previously selected feature is:

$$argmax_{x \in X-Z}(min_{z \in Z}(JMI)); \tag{7}$$

Where Z is the subset of 'best' features selected by JMIM. Note that is for $x \in X - Z$ since the number of features in X decreases by one as each feature 'best' feature is added

to the new subset.

We have also included an example this function with arbitrary values for set jmi for ease of understanding:

0	1	2	3	4
1	2	3	4	5

In this case, index 4 contains the maximum value of 5 in set jmi. This would then be used to return the corresponding feature.

JMIM alike JMI can only be applied to discrete features so continuous features were discretized before its application. It was also necessary that there is a previously selected 'best' feature at each iteration. Since JMIM uses a forward selection search algorithm, it means the subset begins as an empty set, then inserts the feature which performs the best one at a time. The JMIM of the first iteration therefore, cannot be computed with this empty subset of best features since there is no previous value to compute the mutual information of the feature to. To resolve this edge case, we developed the `get_first_fi` function in Figure 4 to return the feature with the highest mutual information with the class label to initialise the subset.

```
def get_first_fi(df, features):
    max_mi = 0
    mi  = 0
    first_fi = 0
    new_features = [] # all features excluding feature w/ max_mi

    for f in features:
        h_x = entropy(df, f)
        h_y = entropy(df, 'HasDetections')
        h_x_y = joint_entropy(df, f, 'HasDetections')
        mi = h_x + h_y - h_x_y

        if mi > min(h_x, h_y) or mi == 0.0:
            continue
        else:
            print(f, mi)
            new_features.append(f)
            if mi > max_mi:
                max_mi = mi
                first_fi = f
    return max_mi, first_fi, new_features
```

Figure 4: Python code snippet of 'get_first_fi' function.

3.4.2 Chi-square test

The Pearson's chi-square test is the second filter method we assessed for comparison. For convenience, we used `chi2` from the `sklearn` Python package to implement the algorithm. It is defined as:

$$\chi^2 = \frac{(O - E)^2}{E} \tag{8}$$

Where:

- O is the frequency of a distinct observed instance.
- E is the expected frequency of distinct observed instance. The expected frequency is computed by multiplying the mean of the class label by the frequency of observed instances.
- χ^2 is Pearson's chi-square test statistic. We recommend the walk-through with an example dataset provided by [77] on the computation of this test statistic.

The Pearson's test statistic values for each frequency of observed and expected instances is returned as an array. The values, along with the number of degrees of freedom, k (the

number of distinct frequency of observed instances) is then input into the Chi-square PDF denoted as:

$$p(x) = \frac{x^{k/2-1}e^{-x/2}}{2^{k/2}\Gamma(k/2)}, \text{ for x} \geq 0. \tag{9}$$

Where:

- x, is an instance of Pearson's test statistics from each distinct observed and corresponding expected frequency.
- Γ, is Euler's gamma function which can only accept real or positive values [78].

This distribution was then used to transform the dataset and the features with corresponded to the k highest values of the PDF were returned.

3.5 Light Gradient Boosting Machine

Light Gradient Boosting Machine (LightGBM) was developed and released by [79] in 2017 as a new alternative to Gradient Boosted Decision Tree (GBDT). This is because GBDT was deemed unsatisfactory for large datasets where the feature dimension was high.

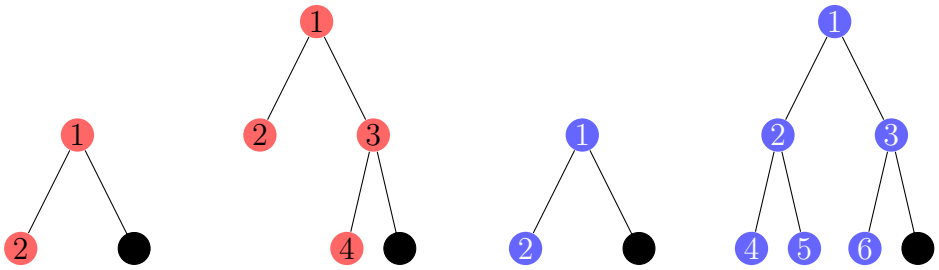

Figure 5: Leaf-wise (red) vs. level-wise (blue) decision tree growth.

Unlike GBDT and other decision tree algorithms such as XGBoost LightGBM grows vertically (leaf-wise) rather than horizontally (level-wise) as illustrated in Figure 5. Level-wise tree generation can be an inefficient process because it treats the leaves at each layer the indiscriminately resulting in unnecessary memory consumption [80]. Instead LightGBM uses a histogram-based algorithm called Exclusive Feature Bundling (EFB) to group similar features into discrete bins. Then the model traverses across each value of each bin until an optimal split point is found. Rather than traverse all the bins again, the optimal split point is used to create 'sub-bins' recursively thus increasing the depth of the tree. In addition, avoid profitless splitting. Without a depth limit, however, this could lead to over-fitting. Hence [79] added a `max_depth` parameter to limit the depth LightGBM can grow to.

3.6 Selection of LightGBM training parameters

The 7 main training parameters which govern the performance of LightGBM are shown in Table 2.

Table 2: Main parameters of LightGBM.

Parameters	***Description***	***Range of values***
`device_type`	The type of logic processing unit the device is run on. It executes instructions from source code line by line. Other units include, the Graphics Processing Unit (GPU) and the Tensor Processing Unit (TPU).	cpu
`num_leaves`	The number of 'leaves' the decision tree should have. The complexity of the model is proportional to the leaves which can improve model accuracy [53].	10, 20, 30, 60, 80
`max_depth`	This limits the depth of the tree which reduces the complexity of the model and thereby reduces over-fitting.	6, 9, 12, 24
`learning_rate`	The rate the model updates its internal model parameters. If the learning rate is too large, the LightGBM may not be able to stabilise. Conversely a learning rate too small, may lead to over-fitting [81].	0.01, 0.05
`lambda_l2`	Formally, L2 regularisation is a technique to reduce over-fitting by limiting the magnitude of the internal weights such that no single feature overwhelms others [34]. For example a value of 0.1 reduces a given weight by 10%.	0.1
`objective`	The type of classification the model is used for. Since there are only two results for the class label, 'HasDetections' it is binary classification.	binary
`metric`	The performance evaluation metric the model is trained against (see Section 3.8).	auc

*Additional parameters such as `verbosity = -1` was used to tell the function to explicitly not to display every combination tried. `random_state = 42` was used to stabilise the search results, otherwise at each run it would return different values for the same combination of parameters.

With these range of parameters we conducted an Exhaustive Grid Search (EGS). For example, given `num_leaves = [20, 30, 50, 80, 100]`, EGS would try each value with different values of other parameters, until the optimal combination of parameters for the LightGBM were found. To accomplish this task we used `GridSearchCV` from the `sklearn` Python package. By default the combination of parameters which produced the best AUC (see Section 3.8) score via cross-validation was the evaluation criterion. To conduct cross-validation the train dataset is split into k subsets called 'folds'. Then starting from 1

to the k^{th} fold, the fold becomes the validation data to assess whether the model has overfitted (train AUC score $\gg$ validation AUC score) while the remaining folds for were used for training the LightGBM. The proportion of training data to validation data is dependent on the size of the dataset. As discussed in Section 2.4, a greater proportion of training data provides more information for the LightGBM to learn from to obtain better results but can also result in a less generalisable statistical model. As commonplace, a 5-fold cross-validation, was used to obtain the parameters [82].

For JMIM and Chi-square, this and the subsequent subsections were repeated for every K optimal features selected as K increased. This is because we cannot guarantee the same combination of parameters for a subset of 5 optimal features ($K = 5$) is also best for 20 features ($K = 20$).

3.7 Training, validation & testing

The dataset is large ($\geq 10^5$) so an 80:20 partition was used for training the final LightGBM model [83]. The final LightGBM that was applied to the test dataset was not trained via cross-validation because it is typically only applicable when the sample size is very small [84]. It would also increase model the time to train by k number of folds. It is important the experiment assesses the predictive capability of the LightGBM so for all models, the control, JMIM and Chi-square, a test dataset independent of the train dataset provided by Microsoft in the same competition was used after the model was trained. Re-substitution, the practice of using the train, validation and test datasets for model training is unacceptable and known to produce unreliable results [85]. Following [28], the models were trained for 1000 iterations. Since the test dataset has no class label we run and compare the similarity of the JMIM with LightGBM and Chi-square with LightGBM on the unlabelled test dataset to the control. The more similar they are the more generalisable our model is likely to be.

3.8 Performance evaluation metrics

Memory usage of the combined control with LightGBM, JMIM with LightGBM and Chi-square with LightGBM were recorded using `%memit` from the `memory_profiler` Python package. Simultaneously the time taken to train the model (execution time) was recorded using the `time` Python package. The results of the binary classification were evaluated using AUC scores, a metric frequently used to rank model performance for binary classification tasks [86], and as done by [1] and [28]. Acceptable AUC score values ranges are defined by [1] - less than 0.5 is unacceptable, 0.5 to 0.7 is acceptable and a good score is above 0.7. The mathematical definition of AUC is presented by [87].

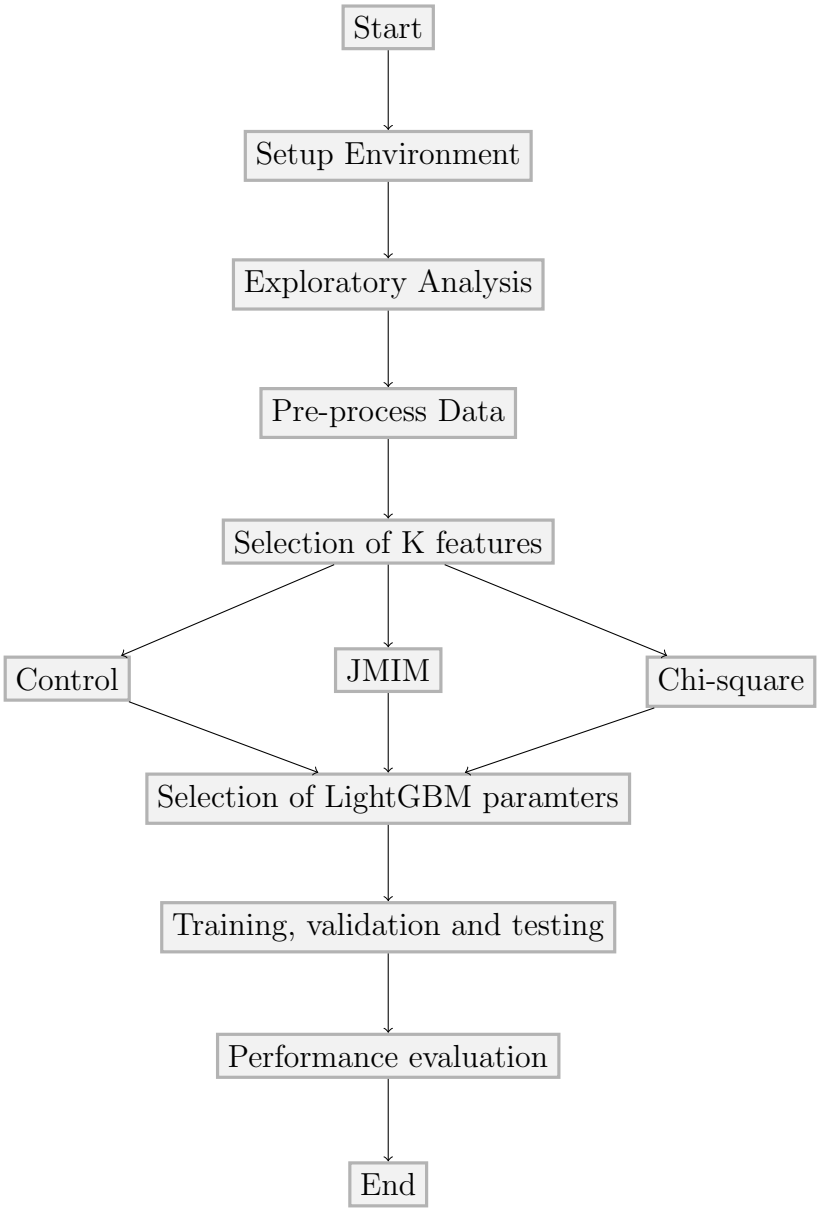

Figure 6: Flowchart of methodology.

4 Results

The first Section 4.1 shows results of the exploratory analysis used to facilitate Section 4.2 onward. Section 4.2 shows the 'best' features selected by JMIM with LightGBM and Chi-square with LightGBM as the size of subset, K, increases. In same section, the average of each performance benchmark, described in Section 3.8, is compared to the control which is a LightGBM trained with all features. Section 4.5 shows results of all three experiments on the testing dataset. Lastly in Section 4.6, a diagnostic check is performed on JMIM with LightGBM and Chi-square with LightGBM memory consumption and execution time results to determine if they were significant.

4.1 Exploratory analysis

4.1.1 Class distribution

Figure 7 shows the number of True and False labels are similar so the train dataset which was necessary to reduce model biases. In some extreme cases of highly imbalanced datasets the minority class label is mistakenly treated as noise and removed [88].

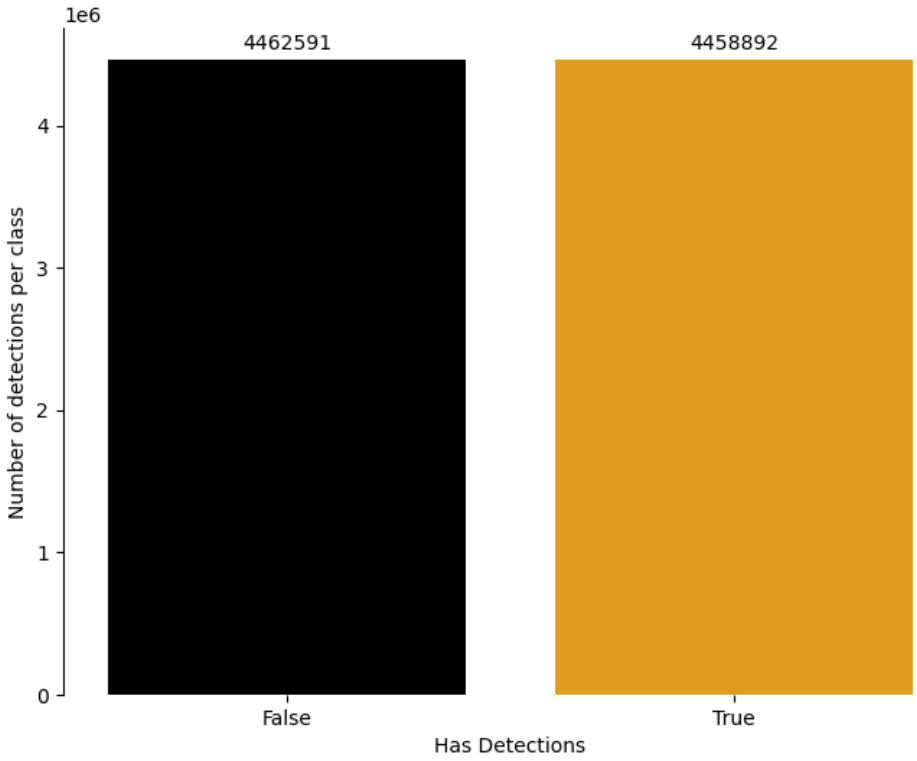

Figure 7: Class label 'HasDetections' distribution.

4.1.2 Proportion of data types

Table 3 shows categorical is the majority data type in the train dataset followed by binary and numerical.

Table 3: Proportion of data types in the train dataset (1.d.p).

Binary	***Categorical****	***Numerical***
24.4%	65.9%	9.7%

*A categorical feature is a feature with greater than two distinct categories otherwise it is binary feature.

4.1.3 Proportion of missing values

Figure 8 shows only 3 features have $\geq 95\%$ proportion of missing instances out of the 82.

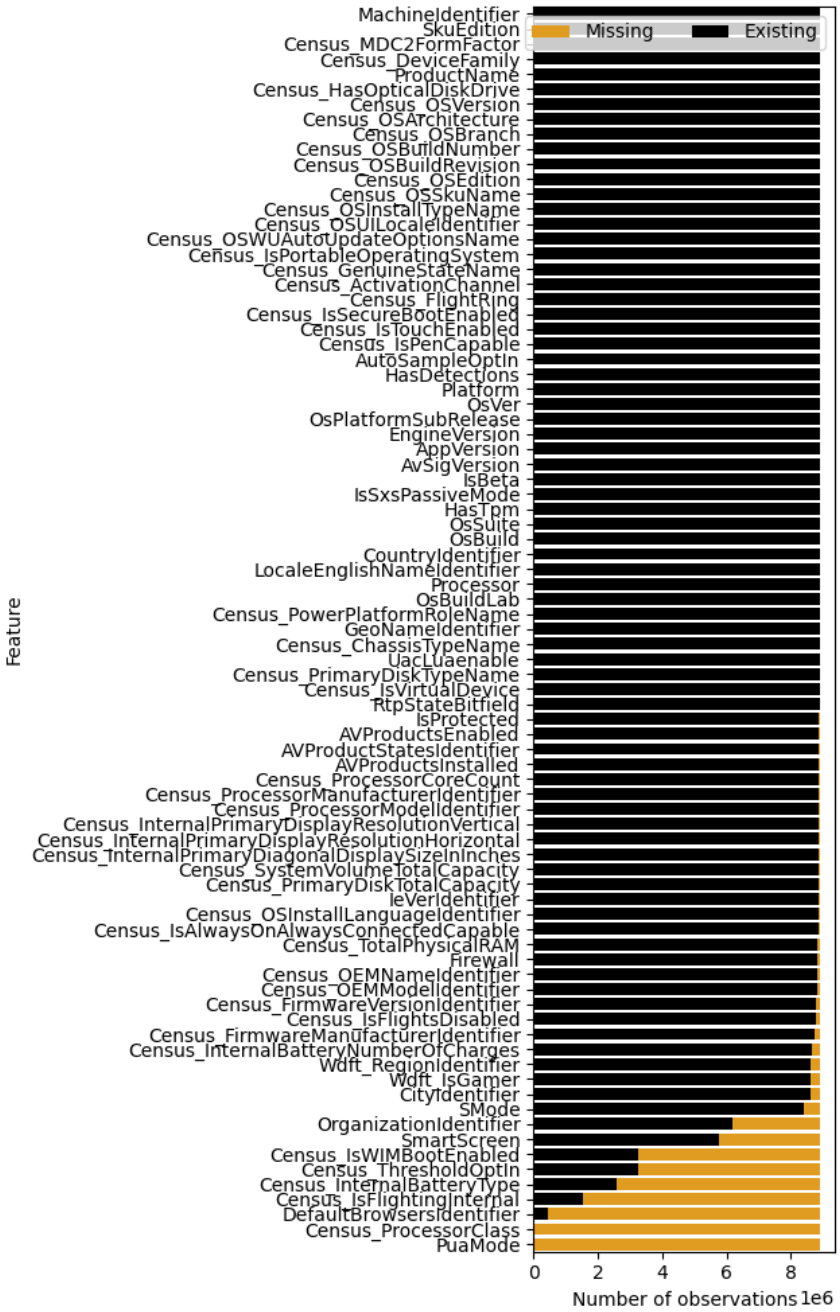

Figure 8: Proportion of missing values of all features in the train dataset.

4.1.4 Cardinality

There were 30 high cardinality features in the dataset. Figure 9 shows 'Census_OEMModelIdentifier' had the most distinct categories, 175365 to be exact, thus had the highest cardinality.

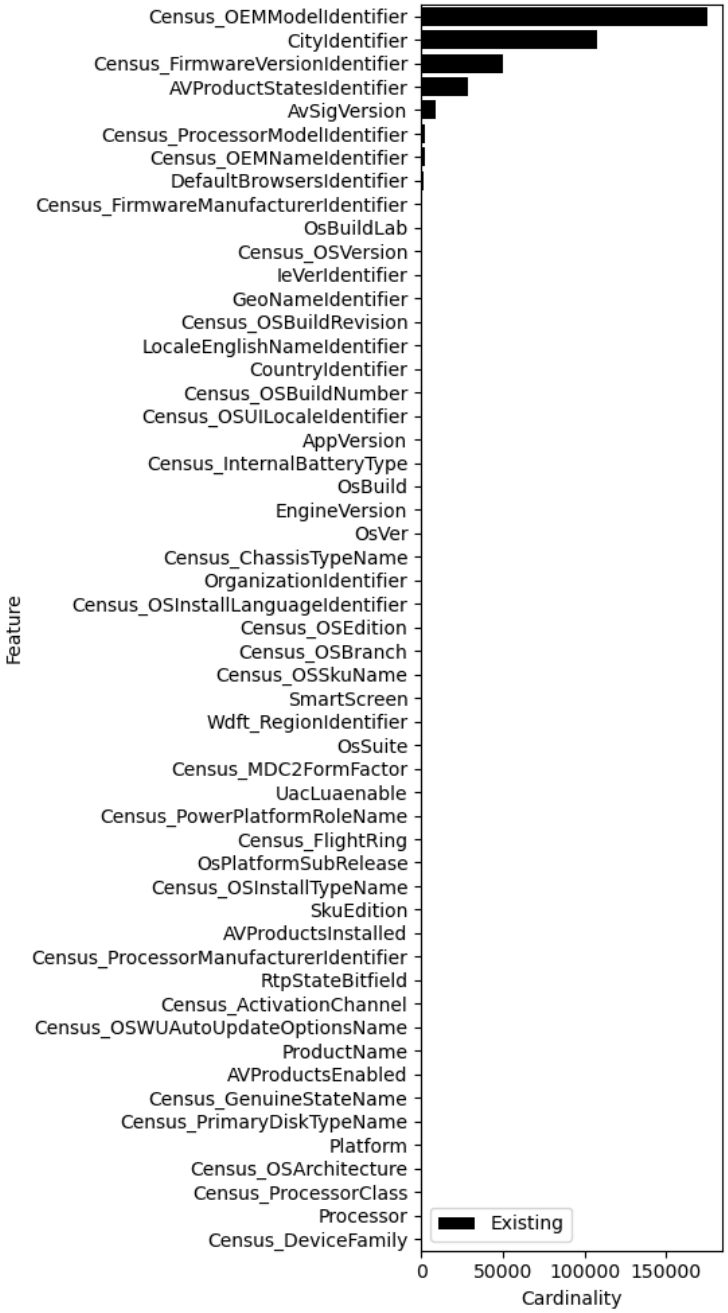

Figure 9: Cardinality of all categorical features in the train dataset.

4.1.5 Linear correlation of numerical features

All Pearson linear correlation coefficients, in Figure 10, are less than 0.7. Most of the numerical features appear to not be collinear except 'CensusInternalPrimaryDisplayResolutionHoriztonal' and 'CensusInternalPrimaryDisplayResolutionVertical' because the linear correlation is 0.9 which is greater than 0.7. This is sensible given the primary display resolution of a machine is always a rectangle where the horizontal length is directly and linearly proportion to the vertical length. Thus if both were selected by any of the filter methods, further investigation would be required to decide which one to remove. Later, Table 8 shows both features were not selected simultaneously for JMIM or Chi-square so further investigation was not required.

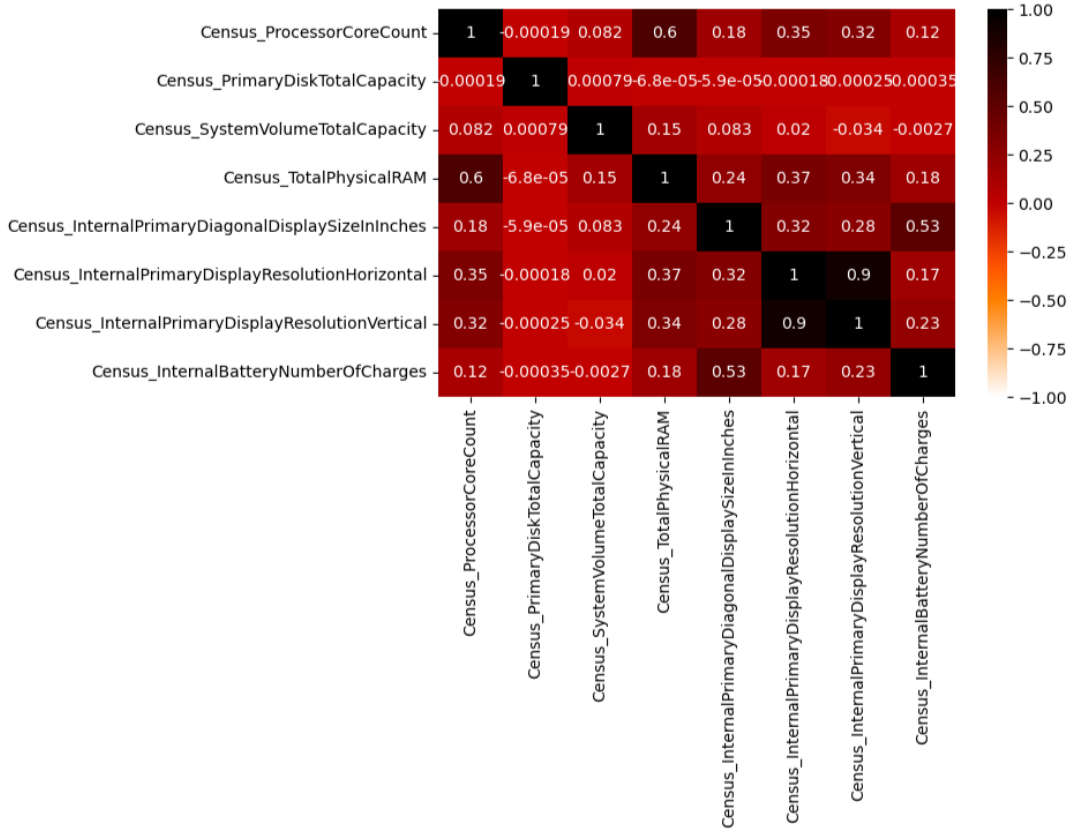

Figure 10: Pearson's linear correlaton between all numerical features.

4.2 LightGBM grid search parameters

Table 4: Optimal LightGBM parameters using Grid Search when $1 \leq K \leq 9$. Where K is the number of elements in the optimal subset of features.

Parameters	***Chosen values***		
	Control	JMIM	CHI
`device_type`	cpu		
`num_leaves`	10	20	10
`max_depth`	6	6	6
`learning_rate`	0.05	0.05	0.05
`lambda_l2`	0.1		
`objective`	binary		
`metric`	auc		

Table 5: Optimal LightGBM parameters using Grid Search when $10 \leq K \leq 14$.

Parameters	***Chosen values***		
	Control	JMIM	CHI
`device_type`	cpu		
`num_leaves`	10	80	80
`max_depth`	6	24	24
`learning_rate`	0.05	0.01	0.05
`lambda_l2`	0.1		
`objective`	binary		
`metric`	auc		

Table 6: Optimal LightGBM parameters using Grid Search when $15 \leq K \leq 19$.

Parameters	***Chosen values***		
	Control	JMIM	CHI
`device_type`	cpu		
`num_leaves`	10	30	80
`max_depth`	6	9	24
`learning_rate`	0.05	0.05	0.05
`lambda_l2`	0.1		
`objective`	binary		
`metric`	auc		

Table 7: Optimal LightGBM parameters using Grid Search when $K = 20$.

Parameters	***Chosen values***		
	Control	JMIM	CHI
`device_type`	cpu		
`num_leaves`	10	30	80
`max_depth`	6	9	24
`learning_rate`	0.05	0.05	0.05
`lambda_l2`	0.1		
`objective`	binary		
`metric`	auc		

4.3 Input feature selection

Table 8: Subset of up to 20 optimal features selected by JMIM and Chi-square. Features which appear in both subsets are highlighted in bold.

Input feature selection method	***Features***
JMIM	'AVProductStatesIdentifier', 'IeVerIdentifier', 'SmartScreen', 'AvSigVersionp2', 'AppVersionp4', **'CensusOSInstallTypeName'**, 'OsBuildLab', 'OsBuild', 'AppVersionp3', 'CensusOSBuildNumber', 'AvSigVersionp3', 'CensusOSVersionp4', 'EngineVersionp3', 'DefaultBrowsersIdentifier', 'OsPlatformSubRelease', 'CensusOSBuildRevision', 'AppVersionp2', 'LocaleEnglishNameIdentifier', 'CensusOSVersionp3', 'CountryIdentifier'
Chi-square	'ProductName', 'RtpStateBitfield', 'AVProductsInstalled', 'AVProductsEnabled', 'Processor', 'OsSuite', 'OsPlatformSubRelease', 'SkuEdition', 'UacLuaenable', 'CensusMDC2FormFactor', 'CensusProcessorManufacturerIdentifier, 'CensusPrimaryDiskTypeName', 'CensusPowerPlatformRoleName', 'CensusOSArchitecture', **'CensusOSInstallTypeName'**, 'CensusOSWUAutoUpdateOptionsName', 'CensusGenuineStateName', 'CensusActivationChannel', 'CensusFlightRing', 'WdftRegionIdentifier'

Table 8 shows the only 'CensusOSInstallTypeName' was selected by JMIM and Chi-square.

4.4 Comparison of filter methods with LightGBM

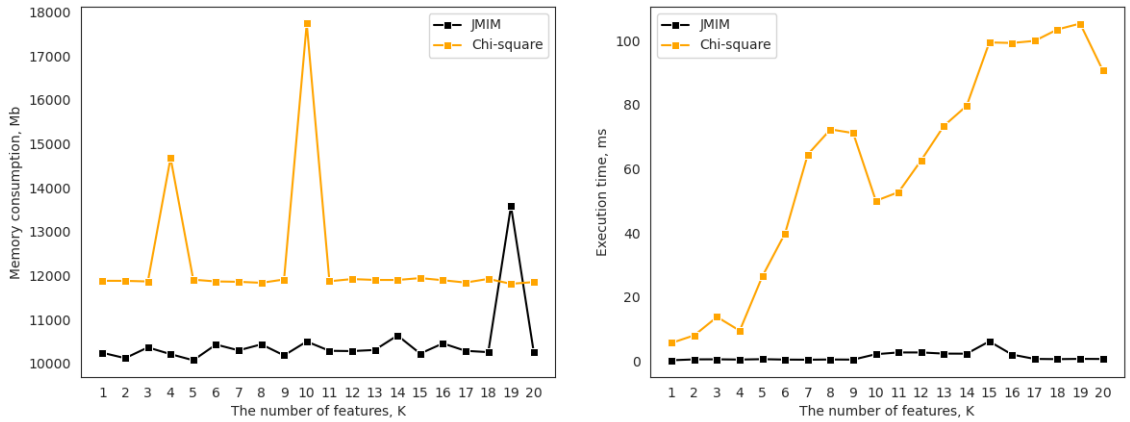

Figure 11: Subset of K features for memory consumption (left) and execution time (right).

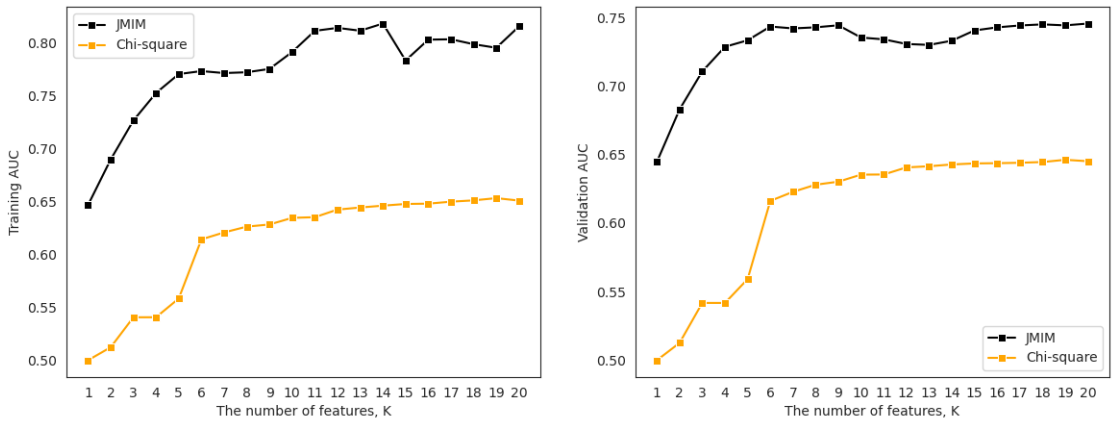

Figure 12: JMIM and Chi-square AUC relative training (left) and validation scores (right).

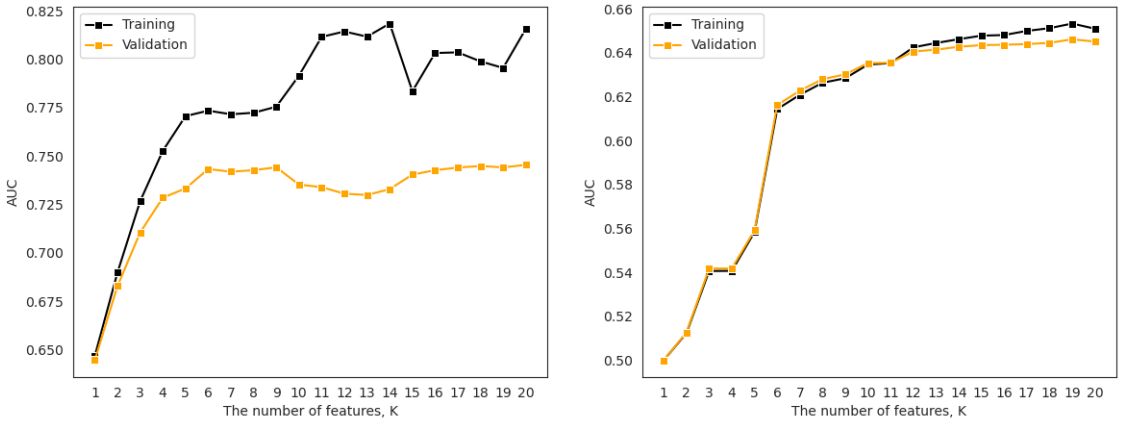

Figure 13: Training and validation JMIM (left) and Chi-square (right) AUC scores.

Table 9: AUC, Execution time (ms) and Memory Consumption (Mb) of JMIM with LightGBM and Chi-square with LightGBM. Where T = Training and V = Validation.

	JMIM				***CHI***			
K	AUC (T)	AUC (V)	Time	Memory	AUC (T)	AUC (V)	Time	Memory
1	0.64680	0.64466	0.31867	10237.96	0.50008	0.50004	5.73630	11879.31
2	0.68988	0.68290	0.50409	10115.91	0.51237	0.51270	8.09440	11877.86
3	0.72673	0.71057	0.61130	10361.20	0.54068	0.54183	13.8219	11864.25
4	0.75275	0.72854	0.52853	10206.99	0.54074	0.54182	9.54135	14679.94
5	0.77062	0.73327	0.65500	10068.11	0.55845	0.55918	26.5271	11904.64
6	0.77343	0.74331	0.52615	10425.99	0.61437	0.61622	39.9189	11864.61
7	0.77157	0.74194	0.49072	10293.65	0.62092	0.62281	64.5074	11856.44
8	0.77234	0.74273	0.55598	10430.71	0.62640	0.62801	72.3206	11833.12
9	0.77547	0.74421	0.51962	10178.39	0.62843	0.63021	71.1701	11909.43
10	0.79150	0.73528	2.20667	10495.32	0.63469	0.63535	50.0432	17742.75
11	0.81150	0.73397	2.75318	10283.71	0.63533	0.63546	52.7447	11868.87
12	0.81427	0.73059	2.74934	10277.09	0.64251	0.64061	62.7108	11918.70
13	0.81161	0.72992	2.39170	10302.71	0.64448	0.64146	73.4340	11898.82
14	0.81837	0.73296	2.36613	10633.02	0.64622	0.64277	79.6604	11898.02
15	0.78345	0.74045	6.20400	10220.63	0.64782	0.64352	99.4446	11940.89
16	0.80316	0.74271	2.08288	10451.52	0.64811	0.64370	99.2625	11892.09
17	0.80351	0.74408	0.73308	10283.11	0.64999	0.64400	99.9302	11839.12
18	0.79885	0.74488	0.66788	10253.32	0.65123	0.64454	103.482	11921.85
19	0.79547	0.74416	0.74908	13582.34	0.65334	0.64617	105.324	11810.89
20	0.81566	0.74548	0.71602	10265.14	0.65101	0.64513	90.7136	11850.03

Input feature selection method	**Average memory consumption (Mb)**	**Average execution time (ms)**	**Average training AUC**	**Average validation AUC**
Control	13582.34	7.4909	0.8106	0.7605
JMIM	10468.36	1.4215	0.7763	0.7298
Chi-square	12312.58	61.4194	0.6124	0.6108

Table 10: Average memory consumption, execution time and AUC scores for Control, JMIM and Chi-square.

Figure 11 (left) illustrates the memory consumption for JMIM with LightGBM is consistently below Chi-square with LightGBM. There is an anomaly when $K = 19$ where the memory consumption of JMIM with LightGBM exceeds Chi-square with LightGBM. One reason could have been the selection of parameters returned from Grid Search were input into the final model parameters incorrectly. If this was the case the model parameters therefore were sub-optimal and diminished results as shown. To mitigate this human error in future experimentation we would feed the results of the Grid Search immediately into the LightGBM model to be trained rather than input it each one manually. This would result in a slower collection of results because the Grid Search would have to be run on every iteration of K from 1 to 20 but with reduced risk of producing anomalous results.

Figure 11 (right) shows Chi-square with LightGBM execution time increases roughly linearly while the JMIM with LightGBM execution time remained relatively constant and below Chi-square with LightGBM. On the contrary, Figure 12 and Figure 13 both filter methods appear to follow a similar trend in absolute and relative training and validation AUC scores where as K increases the AUC scores increase. However, Figure 13 the training and validation scores for JMIM with LightGBM begin to diverge much earlier from $K = 2$ indicative of under-fitting whereas Chi-square with LightGBM which diverges marginally from $K = 12$.

Table 10 shows JMIM with LightGBM shows superior results for average memory consumption and execution time compared to Chi-square with LightGBM and the control. However, we found the control outperformed JMIM with LightGBM in average training AUC and validation AUC.

4.5 'HasDetections' rate on unseen data

The results for the unlabelled test data for JMIM, Chi-square and the control experiment were too big to be included in an Appendix. Therefore, a subset of the first 10 results the results have been included in Table 11 below.

MachineIdentifier	***HasDetections***		
	Control	JMIM	Chi-square
0000010489e3af074adeac69c53e555e	0.533928	0.296639	0.580708
00000176ac758d54827acd545b6315a5	0.430923	0.451965	0.366042
0000019dcefc128c2d4387c1273dae1d	0.458398	0.404254	0.267157
0000055553dc51b1295785415f1a224d	0.565439	0.426865	0.384837
00000574cefffeca83ec8adf9285b2bf	0.546119	0.420212	0.554752
000007ffedd31948f08e6c16da31f6d1	0.412264	0.420054	0.398790
00000a3c447250626dbcc628c9cbc460	0.492711	0.485570	0.332479
00000b6bf217ec9aef0f68d5c6705897	0.324025	0.505959	0.484529
00000b8d3776b13e93ad83676a28e4aa	0.576487	0.413056	0.333203

Table 11: 'HasDetections' rate for the first 10 Windows machines.

Table 11 also shows the neither JMIM with LightGBM and Chi-square with LightGBM test results were the same or exhibited a similar trend to one another or the control. For example, the control LightGBM shows Machine Identifier 0000055553dc51b1295785415f1a224d had the highest 'HasDetections' rate whereas for JMIM with LightGBM and Chi-square with LightGBM the Machine Identifiers were 00000b6bf217ec9aef0f68d5c6705897 and 00000574cefffeca83ec8adf9285b2bf respectively.

4.6 Assessment of the significance of findings

After training, validation and testing, the Wilcoxon test was used to determine whether any differences between the memory consumption and execution time results of the JMIM with LightGBM and Chi-square with LightGBM. The Wilcoxon test is suitable for small samples sizes but the size must be ≥ 16 [89]. The two samples are said to be different if the probability of both having the same distribution is less than 5% (the level of significance). This probability is known as the p-value. Formally defined by [90], a p-value of $< 5\%$ is required for publishable results [91].

Input feature selection method	**p-value**	**Reject null hypothesis? (Y/N)**
Memory consumption	0.0003948211669921875	Y
Execution time	1.9073486328125e-06	Y

Table 12: Wilcoxon test results for JMIM and Chi-square.

Table 12 shows the p-values for JMIM with LightGBM and Chi-sqaure memory consumption and for execution time are significant as both p-values are much less than 0.05. Therefore we reject the null hypothesis which assumes the distributions of the samples are the same.

5 Discussion

5.1 Scalability

The results of our experiment suggest the following remarks. Using the same machine learning model, LightGBM, JMIM consistently in terms of training and validation accuracy and significantly outperformed Chi-square and the control in terms of memory consumption and time taken to train the model (execution time). JMIM with LightGBM generally scales well because as the size of subset increased, the amount of memory consumed and execution time remained relatively constant. This is unlike Chi-square with LightGBM where the execution time increased linearly and ran 5.3 times slower on average. The memory consumption of Chi-square with LightGBM also remained relatively constant but on average used 29% more memory. This disparity in results may be because only one feature was selected by both JMIM and Chi-square whereas all the other features they had selected were unique. Thus, also highlights how fundamentally different the methods are despite both being classed as filter-based input feature selection methods.

5.2 Generalisability

JMIM selected features resulted in a LightGBM model which tended to under-fit the dataset whereas the Chi-square LightGBM model was prone to some over-fitting as the size of the subset of features increased. These observations also provide a possible explanation for the results of testing in Section 4.5. In this section the results showed the 'HasDetections' rate results on the test dataset were dissimilar to the control LightGBM. Unlike the control LighGBM, which uses every feature in the original dataset, JMIM with LightGBM and Chi-square with LightGBM were models which were only trained on the subset features they had selected. This meant more features in the unlabelled test dataset were unseen so both filter-based models may explain why both models found it difficult to generalise.

5.3 Real-world application

Notably, however, neither filter-based method achieved a higher average training or validation accuracy than the control LightGBM. This substantiates the findings of [38] in Section 2.1.3 which found for filter input selection methods there is typically a trade-off of accuracy to increase the speed and/or reduce memory consumption. More broadly, as the difference in average training and validation accuracy of JMIM with LightGBM to the control LightGBM was $< 5\%$ there are some situations where this slight sacrifice could be worthwhile. One example, in line with aim of this study, is when the machine has limited computational resources, such as the smartphone. Smartphones are known to have a plethora of security vulnerabilities from downloading third-party applications or through failure to update the device to the latest software patch [92]. At the same time, the resources of the smartphone are still limited compared to a laptop despite the RAM, CPU and battery capacities of the device having improved significantly over the last few decades [93]. There is constant connectivity so the smartphone also collects and processes large volumes of high dimensional data. Thus, to avoid deprecation of performance users are incentivised to adopt a malware detection method where the additional overhead it introduces is minimal.

6 Conclusions

This study sought to develop a lightweight, machine learning model for machines with limited computational resources. We developed JMIM with LightGBM which consistently outperformed Chi-square with LightGBM in terms of efficiency and scalability with a slight deficit to training and validation accuracy compared to the control. Thus, we recommend JMIM with LightGBM for malware detection on machines with limited computational resources such smartphones.

6.1 Future work and recommendations

This study, however, is not without limitations. To improve generalisability in future work we suggest increasing the size of the subset ($> 25\%$ of the original features) such that less of the features in the test dataset have been unseen by the model. It is important to note the control LightGBM also under-fitted to the dataset. Hence, we also suggest adoption of a dataset with more observations to learn from. To assess the reliability of our conclusion we recommend a repeat of the experiment on a completely separate malware detection dataset featuring machines other than Windows. Other studies such as, [94] sought to develop a hybrid approach to assess whether the combination of the filter and wrapper methods would produce superior results. A comparison of JMIM with the same or different machine learning models to these newer approaches could follow since the original study only compares the hybrid-model to wrapper and machine learning methods without input feature selection.

Bibliography

[1] Ashub Bin Asad, Raiyan Mansur, Safir Zawad, Nahian Evan, and Muhammad Iqbal Hossain. Analysis of Malware Prediction Based on Infection Rate Using Machine Learning Techniques. *2020 IEEE Region 10 Symposium, TENSYMP 2020*, pages 706–709, jun 2020.

[2] Olav Lysne. Static Detection of Malware. *The Huawei and Snowden Questions*, pages 57–66, 2018.

[3] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim. Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey. apr 2011.

[4] Sudhakar and Sushil Kumar. Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. *Future Generation Computer Systems*, 125:334–351, 12 2021.

[5] P. V. Shijo and A. Salim. Integrated Static and Dynamic Analysis for Malware Detection. *Procedia Computer Science*, 46:804–811, jan 2015.

[6] B. Jyothi Kumar, H. Naveen, B. Praveen Kumar, Sai Shyam Sharma, and Jaime Villegas. Logistic regression for polymorphic malware detection using anova f-test. *Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017*, 2018-January:1–5, 1 2018.

[7] Manoun Alazab, Robert Layton, Sitalakshmi Venkataraman, and Paul Watters. Malware detection based on structural and behavioural features of api calls. *International Cyber Resilience conference*, 8 2010.

[8] Pooja Yadav, Neeraj Menon, Vinayakumar Ravi, Sowmya Vishvanathan, and Tuan D. Pham. Efficientnet convolutional neural networks-based android malware detection. *Computers & Security*, 115:102622, 4 2022.

[9] AV-TEST. Malware statistics & trends report — av-test. https://www.av-test.org/en/statistics/malware/, 2023. Accessed: 2023-04-24.

[10] Jagsir Singh and Jaswinder Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, 112:101861, jan 2021.

[11] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. jun 2016.

[12] Qublai K. Ali Mirza, Irfan Awan, and Muhammad Younas. CloudIntell: An intelligent malware detection system. *Future Generation Computer Systems*, 86:1042–1053, sep 2018.

[13] Colin Galen and Robert Steele. Evaluating Performance Maintenance and Deterioration over Time of Machine Learning-based Malware Detection Models on the EMBER PE Dataset. *2020 7th International Conference on Social Network Analysis, Management and Security, SNAMS 2020*, dec 2020.

[14] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. 2 2017.

[15] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita. Mifs-nd: A mutual information-based feature selection method. *Expert Systems with Applications*, 41:6371–6385, 10 2014.

[16] Durmuş Özkan Şahin, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. A novel permission-based android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, 35:4903–4918, 3 2023.

[17] Han Xiao, Claudia Eckert, Chih-Ta Lin, and Nai-Jian Wang. Feature selection and extraction for malware classification. *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING*, 31:965–992, 2015.

[18] Rafiqul Islam, Ronghua Tian, Lynn Batten, and Steve Versteeg. Classification of malware based on string and function feature selection. *Proceedings - 2nd Cybercrime and Trustworthy Computing Workshop, CTC 2010*, pages 9–17, 2010.

[19] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324, 12 1997.

[20] Sebastian B. Thrun, Jerzy W. Bala, Eric Bloedorn, Ivan Bratko, Bojan Cestnik, John Cheng, Kenneth A. De Jong, Saso Dzeroski, Douglas H. Fisher, Scott E. Fahlman, Rainer Hamann, Kenneth A. Kaufman, Stefan Keller, Igor Kononenko, Juergen S. Kreuziger, Ryszard S. Michalski, Tom A. Mitchell, Peter W. Pachowicz, Haleh Vafaie, Walter Van de Welde, Walter Wenzel, Janusz Wnek, and Jianping Zhang. The monk's problems: A performance comparison of different learning algorithms. 1991.

[21] Dirk P. Kroese, Tim Brereton, Thomas Taimre, and Zdravko I. Botev. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6:386–392, 11 2014.

[22] Kai Yao and Jinwu Gao. Law of large numbers for uncertain random variables. *IEEE Transactions on Fuzzy Systems*, 24:615–621, 6 2016.

[23] Jianqing Fan, Weichen Wang, and Ziwei Zhu. A shrinkage principle for heavy-tailed data: High-dimensional robust low-rank matrix recovery. *Annals of statistics*, 49:1239, 6 2021.

[24] Anton A Komar. Single nucleotide polymorphisms methods and protocols second edition. *Methods in Molecular Biology*, 578:3–7, 2009.

[25] Peter Hall and Aurore Delaigle. Effect of heavy tails on ultra high dimensional variable ranking methods. *Statistica Sinica*, 22:909–932, 7 2012.

[26] Waseem Chauhan, Rafat Fatma, Afiya Wahab, and Mohammad Afzal. Cataloging the potential snps (single nucleotide polymorphisms) associated with quantitative traits, viz. bmi (body mass index), iq (intelligence quotient) and bp (blood pressure): an updated review. *Egyptian Journal of Medical Human Genetics*, 23:1–24, 12 2022.

[27] Tingting Chen, Xu Chen, Sisi Zhang, Junwei Zhu, Bixia Tang, Anke Wang, Lili Dong, Zhewen Zhang, Caixia Yu, Yanling Sun, Lianjiang Chi, Huanxin Chen, Shuang Zhai, Yubin Sun, Li Lan, Xin Zhang, Jingfa Xiao, Yiming Bao, Yanqing Wang, Zhang Zhang, and Wenming Zhao. The genome sequence archive family: Toward explosive data growth and diverse data types. *Genomics, Proteomics & Bioinformatics*, 19:578–583, 8 2021.

[28] Kaibo Zhou, Yangxiang Hu, Hao Pan, al, Y Zhong, W Zheng, Z Y Chen, Junchao Liang, Yude Bu, Kefeng Tan, Qiangjian Pan, Weiliang Tang, and Siyue Yao. The Application of LightGBM in Microsoft Malware Detection. *Journal of Physics: Conference Series*, 1684(1):012041, nov 2020.

[29] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection - a comparative study. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4881 LNCS:178–187, 2007.

[30] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N. Cohen, and Adrian Weller. Synthetic data – what, why and how?, 5 2022.

[31] Jonathan A.P. Marpaung, Hoon-Jae Lee, and Mangal Sain. Survey on malware evasion techniques: State of the art and challenges — ieee conference publication — ieee xplore. IEEE, 2 2012.

[32] Giles M. Foody, Ajay Mathur, Carolina Sanchez-Hernandez, and Doreen S. Boyd. Training set size requirements for the classification of a specific class. *Remote Sensing of Environment*, 104:1–14, 9 2006.

[33] Augustin-Louis Cauchy. Analyse mathÉmatique. – méthodc générale pour la résolution des systèmes d'équations simultanées. *Oeuvres complètes*, pages 399–402, 10 2009.

[34] Simukayi Mutasa, Shawn Sun, and Richard Ha. Understanding artificial intelligence based radiology studies: What is overfitting? *Clinical Imaging*, 65:96–99, 9 2020.

[35] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[36] J. M. Sutter and J. H. Kalivas. Comparison of forward selection, backward elimination, and generalized simulated annealing for variable selection. *Microchemical Journal*, 47:60–66, 2 1993.

[37] Yuchun Tang, Yan-Qing Zhang, and Zhen Huang. Development of two-stage svm-rfe gene selection strategy for microarray expression data analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4:365–381, 7 2008.

[38] Sebastián Maldonado and Richard Weber. A wrapper method for feature selection using support vector machines. *Information Sciences*, 179:2208–2217, 6 2009.

[39] Mario Bunge. A general black box theory. *Philosophy of Science*, 30:346–358, 10 1963.

[40] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 51, 8 2018.

[41] Alex A. Freitas. Comprehensible classification models. *ACM SIGKDD Explorations Newsletter*, 15:1–10, 3 2014.

[42] A. Jović, K. Brkić, and N. Bogunović. A review of feature selection methods with applications. *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015 - Proceedings*, pages 1200–1205, 7 2015.

[43] Chaouki Khammassi and Saoussen Krichen. A ga-lr wrapper approach for feature selection in network intrusion detection. *Computers & Security*, 70:255–277, 9 2017.

[44] Geeta Kocher and Gulshan Kumar. Analysis of machine learning algorithms with feature selection for intrusion detection using unsw-nb15 dataset. *International Journal of Network Security & Its Applications (IJNSA)*, 13, 2021.

[45] Farzeen Zehra, Maha Javed, Darakhshan Khan, and Maria Pasha. Comparative analysis of c++ and python in terms of memory and time. *Preprints*, 12 2020.

[46] Alper Sarıkaya and Banu Günel Kılıç. A class-specific intrusion detection model: Hierarchical multi-class ids model. *SN Computer Science*, 1:1–11, 7 2020.

[47] Viv Bewick, Liz Cheek, and Jonathan Ball. Statistics review 8: Qualitative data - Tests of association. *Critical Care*, 8(1):46–53, feb 2004.

[48] Mohamed Bennasar, Yulia Hicks, and Rossitza Setchi. Feature selection using Joint Mutual Information Maximisation. *Expert Systems with Applications*, 42(22):8520–8532, dec 2015.

[49] Omar A.M. Salem, Feng Liu, Yi Ping Phoebe Chen, and Xi Chen. Feature selection and threshold method based on fuzzy joint mutual information. *International Journal of Approximate Reasoning*, 132:107–126, 5 2021.

[50] Juan C. Pichel and Beatriz Pateiro-Lopez. A New Approach for Sparse Matrix Classification Based on Deep Learning Techniques. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, 2018-September:46–54, oct 2018.

[51] Zhida Li, Ana Laura Gonzalez Rios, and Ljiljana Trajkovic. Machine Learning for Detecting the WestRock Ransomware Attack using BGP Routing Records. *IEEE Communications Magazine*, mar 2022.

[52] Sreelekshmy Selvin, R. Vinayakumar, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. Stock price prediction using LSTM, RNN and CNN-sliding window model. *2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2017*, 2017-January:1643–1647, nov 2017.

[53] Jin Zhang, Daniel Mucs, Ulf Norinder, and Fredrik Svensson. Lightgbm: An effective and scalable algorithm for prediction of chemical toxicity-application to the tox21 and mutagenicity data sets. *Journal of Chemical Information and Modeling*, 2019.

[54] Robert P. Sheridan, Andy Liaw, and Matthew Tudor. Light gradient boosting machine as a regression method for quantitative structure-activity relationships. 4 2021.

[55] Sungwoo Park, Seungmin Jung, Seungwon Jung, Seungmin Rho, and Eenjun Hwang. Sliding window-based lightgbm model for electric load forecasting using anomaly repair. *Journal of Supercomputing*, 77:12857–12878, 11 2021.

[56] Muhamad Erza Aminanto and Kwangjo Kim. Improving detection of wi-fi impersonation by fully unsupervised deep learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10763 LNCS:212–223, 2018.

[57] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal components analysis to the exponential family. *Advances in Neural Information Processing Systems*, 14, 2001.

[58] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. *Springer Topics in Signal Processing*, 2:1–4, 2009.

[59] Carsten F. Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel Carré, Jaime R.García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J. Leitão, Tamara Münkemüller, Colin Mcclean, Patrick E. Osborne, Björn Reineking, Boris Schröder, Andrew K. Skidmore, Damaris Zurell, and Sven Lautenbach. Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. *Ecography*, 36:27–46, 1 2013.

[60] Jamal I. Daoud. Multicollinearity and regression analysis. *Journal of Physics: Conference Series*, 949:012009, 12 2017.

[61] Jason W. Osborne and Elaine Waters. Multiple regression assumptions, 8 2002.

[62] Haofeng Wang, Kai Qi, Yin Tang -, Ma Xin, Huang Xin, Lu Li, al, Chang Liu, Liu Yang, and Jingyi Qu. A structured data preprocessing method based on hybrid encoding. *Journal of Physics: Conference Series*, 1738(1):012060, jan 2021.

[63] Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, 37(5):2671–2692, nov 2022.

[64] Jobin Wilson, Amit Kumar Meher, Bivin Vinodkumar Bindu, Santanu Chaudhury, Brejesh Lall, Manoj Sharma, and Vishakha Pareek. Automatically optimized gradient boosting trees for classifying large volume high cardinality data streams under concept drift. pages 317–335, 2020.

[65] Yoojin Kwon, Michelle Lemieux, Jill McTavish, and Nadine Wathen. Identifying and removing duplicate records from systematic review searches. *Journal of the Medical Library Association : JMLA*, 103:184, 10 2015.

[66] Prem Melville, Nishit Shah, Lilyana Mihalkova, and Raymond J. Mooney. Experiments on ensembles with missing and noisy data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3077:293–302, 2004.

[67] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, 69:16, 6 2004.

[68] David J C Mackay. *Information Theory, Inference, and Neural Networks*. Cambridge University Press, 2005.

[69] Pieter Adriaans. Learning and the cooperative computational universe. *Philosophy of Information*, pages 133–167, 1 2008.

[70] Nojun Kwak and Chong Ho Choi. Input feature selection for classification problems. *IEEE Transactions on Neural Networks*, 13:143–159, 1 2002.

[71] Mai Vu. Lecture 1: Entropy and mutual information. *EE194 – Network Information Theory*, 2013.

[72] C E Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:623–656.

[73] Anthony J Bell and Terrence J Sejnowski. A non-linear information maximisation algorithm that performs blind separation. *Advances in Neural Information Processing Systems*, 7, 1994.

[74] Nguyen Xuan Vinh, Shuo Zhou, Jeffrey Chan, and James Bailey. Can high-order dependencies improve mutual information based feature selection? *Pattern Recognition*, 53:46–58, 5 2016.

[75] Howard Hua Yang and John Moody. Feature selection based on joint mutual information. *Computational Intelligence Methods and Applications*, 1999.

[76] Gavin Brown and Adam Pocock. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection ming-jie zhao mikel luján. *Journal of Machine Learning Research*, 13:27–66, 2012.

[77] Nikolaos Pandis. The chi-square test. *American Journal of Orthodontics and Dentofacial Orthopedics*, 150:898–899, 11 2016.

[78] G. D. Anderson and S.-L. Qiu. A monotoneity property of the gamma function. *Proceedings of the American Mathematical Society*, 125:3355–3362, 1997.

[79] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. 2017.

[80] Vikrant A. Dev and Mario R. Eden. Formation lithology classification using scalable gradient boosted decision trees. *Computers & Chemical Engineering*, 128:392–404, 9 2019.

[81] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. 3 2018.

[82] Li Yang and Junlin Liu. Tuningmalconv: Malware detection with not just raw bytes. *IEEE Access*, 8:140915–140922, 2020.

[83] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. *Departmental Technical Reports (CS)*, 2 2018.

[84] Kentaro Ito and Ryohei Nakano. Optimizing support vector regression hyperparameters based on cross-validation. *Proceedings of the International Joint Conference on Neural Networks*, 3:2077–2082, 2003.

[85] Nathalie Japkowicz and Mohak Shah. Performance evaluation in machine learning. *Machine Learning in Radiation Oncology*, pages 41–56, 2015.

[86] Shaomin Wu, Peter Flach, and Cèsar Ferri. An improved model selection heuristic for AUC. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4701 LNAI:478–489, 2007.

[87] Toon Calders and Szymon Jaroszewicz. Efficient auc optimization for classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4702 LNAI:42–53, 2007.

[88] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 3201:39–50, 2004.

[89] Alok Kumar Dwivedi, Indika Mallawaarachchi, and Luis A Alvarado. Analysis of small sample size studies using nonparametric bootstrap test with pooled resampling method. 2017.

[90] David Colquhoun. The reproducibility of research and the misinterpretation of p-values. *Royal Society Open Science*, 4, 12 2017.

[91] Chittaranjan Andrade. The p value and statistical significance: Misunderstandings, explanations, challenges, and alternatives. *Indian Journal of Psychological Medicine*, 41:210–215, 5 2019.

[92] Milad Taleby Ahvanooey, Qianmu Li, Mahdi Rabbani, and Ahmed Raza Rajput. A survey on smartphones security: Software vulnerabilities, malware, and attacks. *International Journal of Advanced Computer Science and Applications*, 8, 1 2020.

[93] Ping Yan and Zheng Yan. A survey on dynamic mobile malware detection. *Software Quality Journal*, 26:891–919, 9 2018.

[94] S Kalaivani and G Gopinath. Modified bee colony with bacterial foraging optimization based hybrid feature selection technique for intrusion detection system classifier model. *ICTACT Journal on Soft Computing*, 10:2146–2152, 2020.