Decision Trees and Unbalanced Data Sets: An Investigation

Daniel Jones, Junfan Huang

January 2, 2022

Introduction

In this task, we aim to develop model which can infer the specific type of malicious network traffic within the KDD-99 data set [4]. Further, we also hope to gain a deeper insight into the data set. Decision trees (and related models) provide a clear, easily visualised insight into how features in the data set relate to their class labels. For this reason we will focus on models based upon a decision tree.

We will start with a baseline model of a decision tree, then determine it's weak-points. Next, we will investigate methods of addressing those weak-points, and hope to produce a derived model with increased performance. We will then evaluate how these models interact with the data.

Firstly, we implement and inspect a decision tree model. We look at the variable importance of each feature, and compare it with the models performance, looking for potential optimisations. We attempt to isolate specific types of attacks which the baseline model struggles to distinguish.

The concept of model stacking is then applied, using a decision tree as the meta-model. By doing this, we can combine models with a diverse set of aims together to complete the final goal.

Next, we look at methods of increasing decision tree performance using random forests, which apply the concept of classifier boosting to decision trees.

Finally, we investigate methods of balancing the data set in order to improve our models performance. In particular, we use the SMOTE[3] method for oversampling rare classes using synthesized data points.

The code for this project can be found at `http://github.com/dj311/data-science-toolbox-3`:

- `./project/report.Rmd` contains the code for running our simulations and producing the figures in this report.
- `./documentation/` contains rough versions of these and provides evidence of our work.

We have chosen to split the equity as follows:

- Junfan Huang: 50%
- Daniel Jones: 50%

Decision Trees

A decision tree uses the model of an n-ary tree (often binary in practice) to represent a series of decisions, each of which is used to narrow down the set of potential choices. Each non-leaf node represents a decision, and each leaf represent a choice or classification.

By starting at the root node with a single data point and following each decision node, the data point is given the classification of the leaf node it arrives at. A binary decision tree provides a method of splitting the search space in two at each decision point, allowing us to perform a binary search. The algorithm for making a decision on a balanced binary decision therefore needs to make only $\mathcal{O}(log_2(\text{number of leaves}))$.

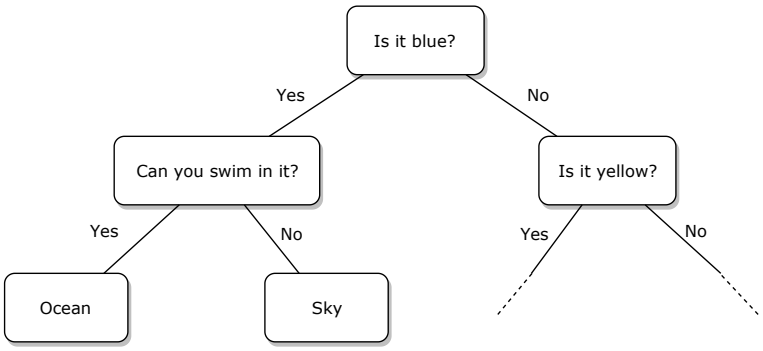

Figure 1: Example of a binary decision tree. If you answer "Yes" to the first two questions, the classification will be "Ocean".

In the context of machine learning, the problem of creating this tree and it's decision points is left to the computer. An optimisation procedure is used at each point to determine the choice which will most effectively split the data set into two distinct categories.

In this project we have used the `rpart` library's implementation of decisions trees, based upon the ideas in the CART (Classification and Regression Trees) book [2]. A measure of *information gain* is used to determine the best value to split the data set on for each feature. The feature with the most effective split is then selected as a decision point. This procedure is repeated recursively, and in a greedy manner. Two common metrics used are:

1. *Gini Impurity* is a measure of how often a randomly chosen element from the resulting set would be given an incorrect label, assuming the remaining labels are distributed uniformly. It is defined as:
$$Gini(f) = 1 - \sum_{c=1}^{C} p_{f,c}^2$$
2. *Entropy*, a measure of information, where we aim to minimise the information on each side of the split (maximising the information gained by making the split). It is defined as:
$$H(f) = -\sum_{c=1}^{C} p_{f,c} \log p_{f,c}$$

where f is the feature, and and $c \in C$ is the class.

Naively applying the `rpart` algorithm (with 10-fold cross-validation) resulted in the model predicting only three classes over the whole data set (Figure 2). The entropy measure was used to determine its splits.

Predicted	True	Count
normal.	back.	2203
neptune.	satan.	1500
normal.	ipsweep.	1243
normal.	warezclient.	1020
normal.	neptune.	973
normal.	portsweep.	940
normal.	teardrop.	931
neptune.	normal.	433
smurf.	normal.	345
smurf.	pod.	259
...	...	...
	Total	10503

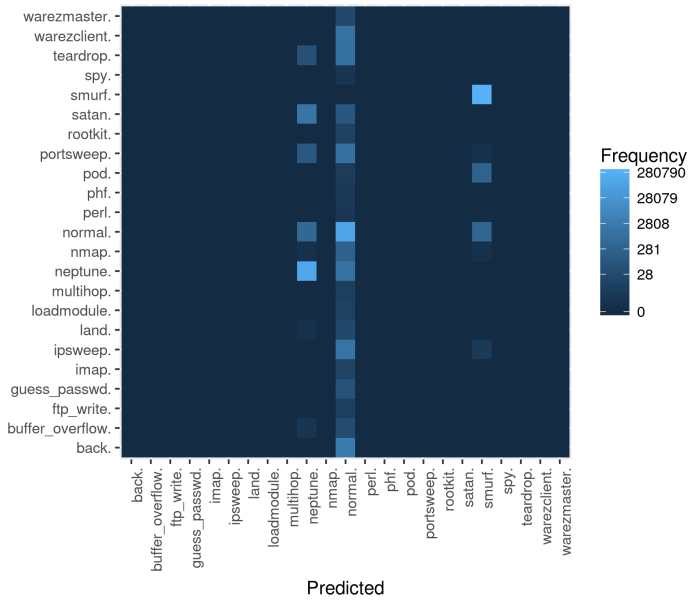

Figure 2: Results from direct application of the `rpart` algorithm. The colouring is scaled logarithmically since the number of classification errors is small compared to the number of successes.

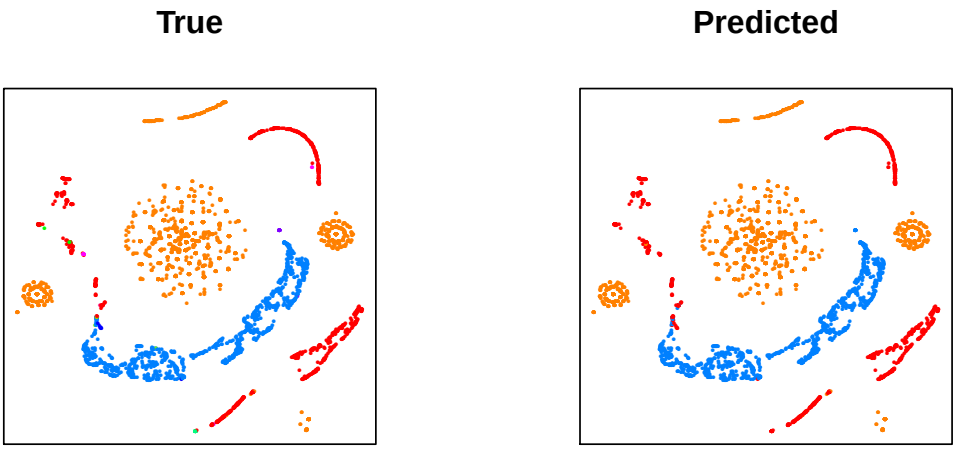

Figure 3: The performance of decision tree

From this figure above, we can find the decision tree made around 10,000 mistakes for achieving our inference goals with small size of testing data. We want to reduce these errors as much as possible.

Investigating Areas for Improvement

In this section we investigate the performance of the pure decision tree, and discuss a number of potential improvements.

Due to the way decision trees measure information gain, a decision which is able to distinguish a between

a large number of rows will be considered more important. This is a good thing, but can cause situations like in Figure 2, where the decision tree *only* ever predicts three of the traffic classes! Here, the decision tree chose to optimise for the most common classes: `smurf`, `neptune` and `normal`. Each of which consists of 56%, 22% and 20% of the data set (respectively) as seen in Figure 1.

Intuitively, it makes sense that `neptune.` and `smurf.` attacks dominate the data set: they are both denial-of-service attacks, and are best characterized by flooding the network with large amounts of data! This is in contrast to one of the rarer attack classes, such as 'phf' which is a targeted attack on a particular version of the Apache web server which exploits a character escaping bug.

This shows that a direct application of the decision tree can classify the DOS classes accurately, but what about the other classes?

Attack	smurf.	neptune.	normal.	back.	satan.	ipsweep.	portsweep.	warezclient.
Frequency	280790	107201	97277	2203	1589	1247	1040	1020

Attack	teardrop.	pod.	nmap.	guess_passwd.	buffer_overflow.	land.	warezmaster.
Frequency	979	264	231	53	30	21	20

Attack	imap.	rootkit.	loadmodule.	ftp_write.	multihop.	phf.	perl.	spy.
Frequency	12	10	9	8	7	4	3	2

Table 1: A table showing the frequency of each attack in the `kddcup.data_10_percent.gz` data set.

Prioritising Rare Classes in Decision Trees

In this section, we add penalty for misclassification of rare classes, and consider how this affects the performance of the model. We use the following formula to weight each class:

$$Penalty(c) = \frac{1}{Percentage of rows of class c}$$

This function grows faster as the rareness of a class increases, we think this is important to ensure that even classes with 2 data points in the training set are given a leaf node in the final decision tree. Looking at the results of this in Figure 4 it can be seen that the weighted decision tree performs far worse. This makes sense, because the penalty for misclassifying datapoints as a rare class is so high, it predicts thousands of `normal` data points as these rare classes!

Predicted	True	Count
teardrop.	normal.	7776
perl.	normal.	5117
guess_passwd.	normal.	4747
warezclient.	normal.	4155
ftp_write.	normal.	1761
rootkit.	normal.	1557
pod.	normal.	1269
loadmodule.	normal.	972
spy.	normal.	860
imap.	normal.	517
...	...	...
	Total	32117

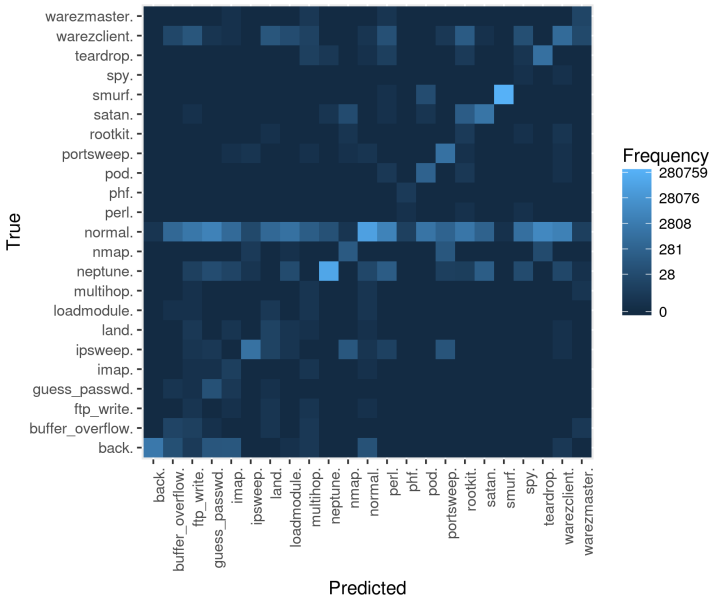

Figure 4: Results from the `rpart` algorithm when class weightings has been used to prioritise rare classes. The colouring is scaled logarithmically since the number of classification errors is small compared to the number of successes.

Exploration for features

Create new features

In KDD-99 data set, there are 41 features for each connection, but these features can not represent a particular event well. For example, there are 4 types of attacks and normal connections and their data might be easy to be recognised by our model, can we use some methods to create new features to fit these four types of attacks? But how to get 5 classes? K-means might be a good choice. Although it is an unsupervised learning algorithm, it is good at finding the different clusters which sounds suitable for our issue. So we firstly tried to implement k-means and got 5 classes. However, it didn't give us a satisfactory result in spite of dividing the attacks quite well with only these five classes. Some DOS events like smurf and neptune are divided into different parts. Although k-means results didn't help us for dividing data into four types of attacks, we got something new at least. If we increase the number of k, we deserve more. This method is what we used for create new features. After generating these features, we are able to compare their importance in the decision tree with the variable importance and some of them might be as a good feature.

Feature selection

In contrast, feature selection is another interesting scheme while exploring the features especially for a tree model. In addition, obtaining the variable importance is not hard by using r packages. Also, we are curious about why some attacks could be predicted well with few data. One of the reasons should be there are some features are super important and helpful for their identifications, like land attacks. Because of these reasons, we tried to find the top 10 important feature for the Kdd-99 data set. Firstly, when we scanned the confusion matrix, we found that there are some attacks except smurf and neptune have 100% predicting accuracy, namely back, guess passwd, imap, land, loadmodule, perl, phf, pod, spy and teardrop. All the DOS attacks obtained a high score for predicting, which means that they might have some good features in the dataset. If we look through the visualisation graph, the smurf data is basically like circles stand out from other connections.

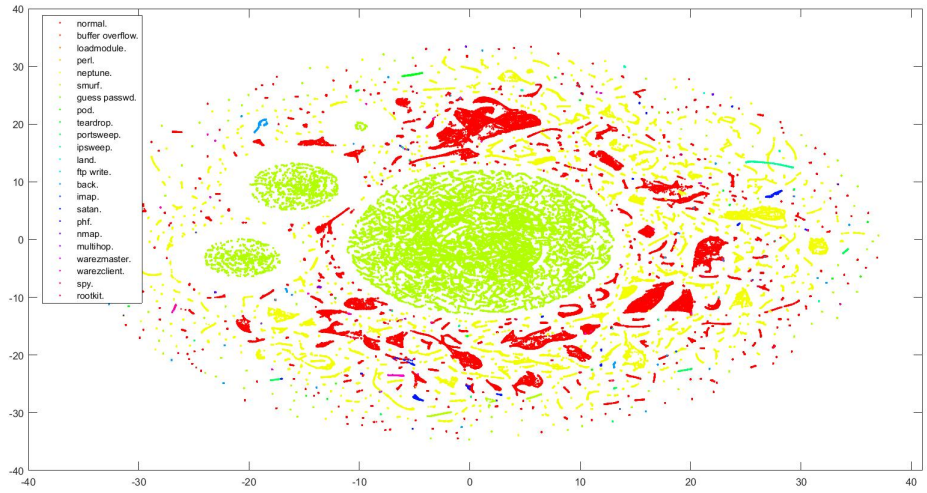

Figure 5: TSNE method for visualization

Figure 6 shows the DOS attacks' distribution (randomly choose 40000 rows for visualisation).

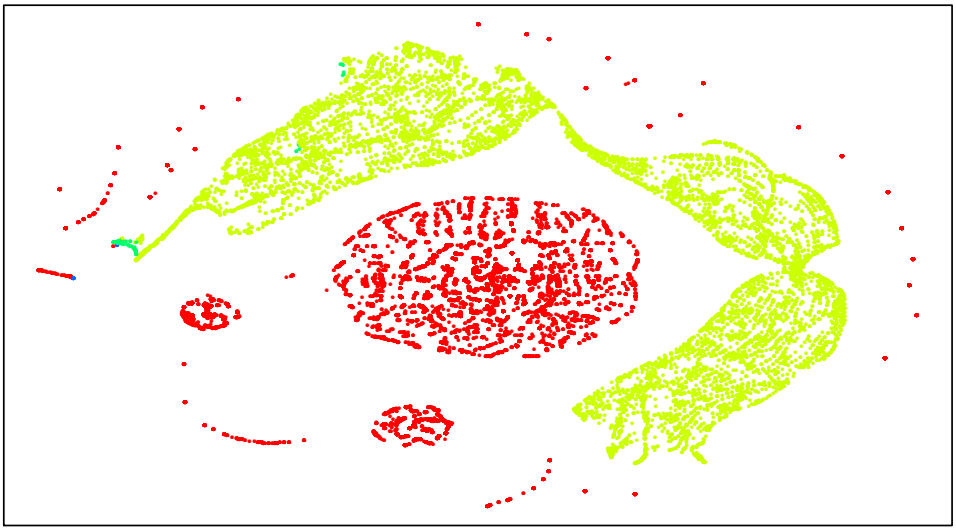

Figure 6: DOS attacks: red is smurf, green is neptune

Embedding Specialised Classifiers within Decision Trees

From our understanding of decision trees, features which can be split in a way that *maximises information gain* will tend to be selected as decision points in the tree. In this section, we consider the possibility

of using predictions from other classifiers as input features into the decisions tree. If these new features provide more information than existing features, they should be used as decision points by the decision tree. Alternatively, this could be interpreted as stacking the the embedded classifiers using a decision tree as the meta-learning algorithm.

Predicted	True	Count
`teardrop.`	`normal.`	7776
`perl.`	`normal.`	5117
`guess_passwd.`	`normal.`	4747
`warezclient.`	`normal.`	4155
`ftp_write.`	`normal.`	1611
`rootkit.`	`normal.`	1557
`pod.`	`normal.`	1269
`imap.`	`normal.`	895
`spy.`	`normal.`	860
`loadmodule.`	`normal.`	783
...	...	...
	Total	32121

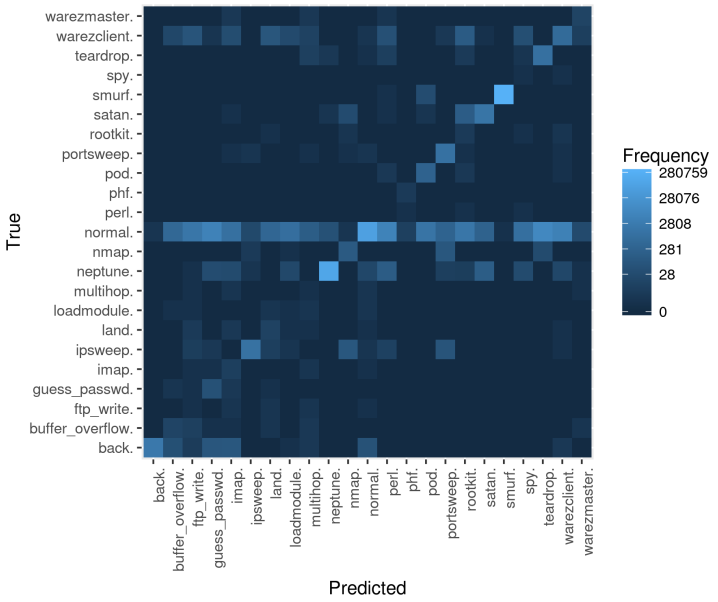

Figure 7: Results from the `rpart` algorithm with an addition feature derived from running k-means with 2 clusters on the training data set. The colouring is scaled logarithmically since the number of classification errors is small compared to the number of successes.

We believe this approach has potential advantages over stacking techniques such as *majority voting*. With majority voting, each embedded classifier has a constant weighting for each classification. This means that weaker classifiers will negatively affect all results. In contrast, using a decision tree allows a more nuanced application of the embedded predictions. For example, it might be the case that one classifier performs well at distinguishing network scanning attacks (`probe`) from denial-of-service (`dos`) attacks, but performs poorly on all other rows of the data. A decision tree allows this classification to be used solely for the purpose of distinguishing *probe* and *dos* attacks. Further, it is possible for the decision tree to filter out poor classifiers completely (this can be seen in our embedding of a 2-cluster k-means classifier later on).

In summary, this technique uses a decision tree to rank classifiers on subsets of the data using it's information gain metric (entropy or gini impurity), and to only apply the classifications for the best ranking classifier on that subset. This idea is based on a Meta Decision Tree (MDT) was developed by Todorovski and Dzeroski [5], which uses a decision tree to decide which classifier apply to the data set. It was later shown [7] to perform better than bagged and boosted decision trees on the twenty-one data sets used.

From the analysis in the above section, it is clear that the majority of the decision trees errors come from classifying malicious traffic as non-malicious (`normal`). Can we embed a classifier that focuses on distinguishing malicious and non-malicious traffic? And if so, will this offset the issues caused by weighting. Figure 7 shows the results of embedding a k-means clustering algorithm on two clusters within the weighted decision tree. It hasn't helped particulary well, in fact, inspecting the model shows that it does not even take into account this new feature! We had slightly better results when trying to find 5 clusters (one for each of the broader attack categories). It's results are shown in Figure 11.

Predicted	True	Count
teardrop.	normal.	7776
guess_passwd.	normal.	5252
perl.	normal.	5117
warezclient.	normal.	4295
ftp_write.	normal.	2073
rootkit.	normal.	1557
pod.	normal.	1269
spy.	normal.	860
loadmodule.	normal.	850
imap.	normal.	436
...	...	...
	Total	33010

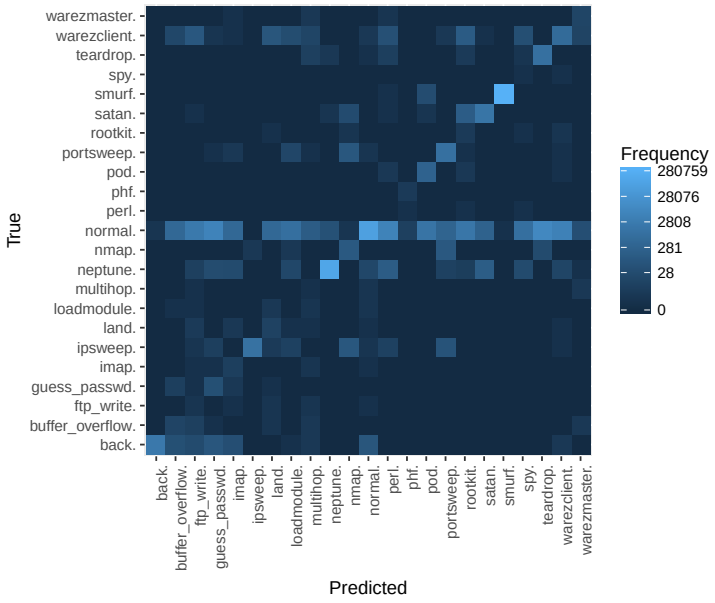

Figure 8: Results from the `rpart` algorithm with an additional feature derived from running k-means with 5 clusters on the training data set. The colouring is scaled logarithmically since the number of classification errors is small compared to the number of successes.

Figure 9: A diagram showing the decision tree with the k-means 5 clustering embedded. You can see how the `k_means_cluster` feature has been used to identify `guess_passwd` attacks. This rendering isn't the clearest, apologies to the reader.

Although K-means clustering technique can create something new, it didn't take too much improvement on the performance whether we added more features or just used these 5 or 20 features we created for training. K-means is a distance based model might be a reason when decision tree doesn't. Since this method failed, we had to find another way.

Random Forests

A decision tree is weak, how about tons of them? The answer is random forests. Random Forest algorithm is a classification algorithm based on ensemble learning. An ensemble-learning model works by aggregating multiple Machine Learning models to reduce error and improve performance. Each of the models (the decision trees), when used on their own, is bad. However, when used together in an ensemble (random forest), the models are strong, and therefore deserve more accurate results.

A random forest trains each decision tree with a different subset of training data. Each node of each decision tree is split using a randomly selected attribute from the data. This element of randomness ensures that the Machine Learning algorithm creates models that are not correlated with one another. As a result, potential errors are evenly spread throughout the model and are cancelled out by the majority voting decision strategy of the model. [1]

From these figure below, it is easy to see that after implementing random forests model using the same data, however, these errors disappeared.

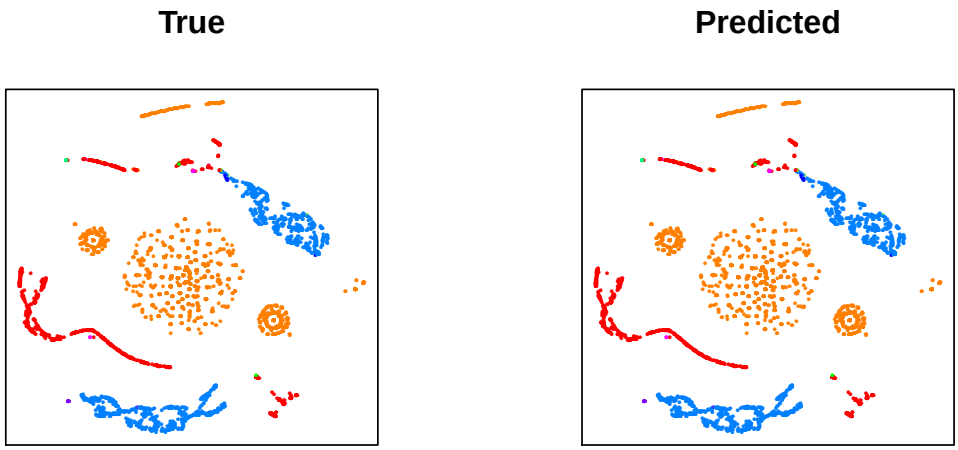

Figure 10: Random forest: "boosted decision trees"

Predicted	True	Count
nmap.	ipsweep.	49
warezclient.	normal.	12
satan.	normal.	11
rootkit.	normal.	10
nmap.	normal.	9
normal.	warez client.	8
buffer_ overflow.	normal.	6
ftp_write.	normal.	5
ipsweep.	normal.	5
loadmodule.	normal.	4
...	...	...
	Total	161

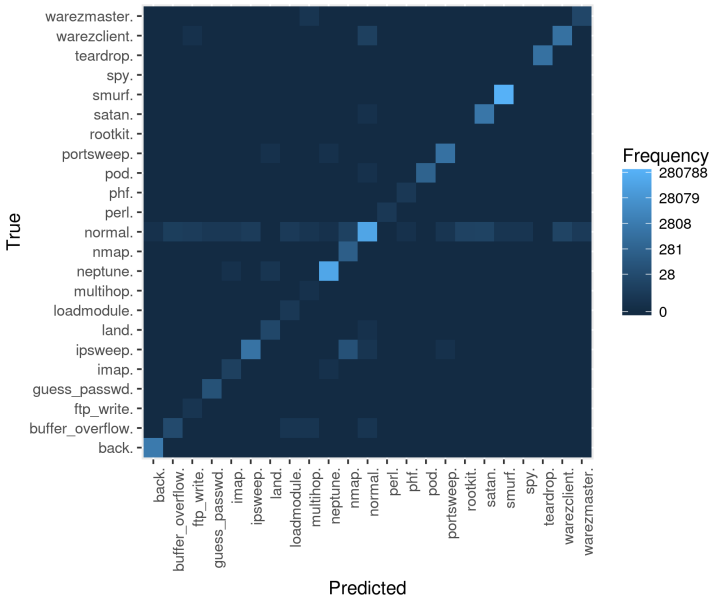

Figure 11: Results from the `ranger` libraries random forest implementation. It has been trained on all features, using 10-fold cross validation. The colouring is scaled logarithmically since the number of classification errors is small compared to the number of successes.

Improving Random Forest performance on Rare Classes

Whilst it is clear that a random forest model performs very well with our data set, it is hard to analyse. Using the default settings, the `ranger` library has developed a random forest of five-hundred trees!

Unbalanced Data

We can try to solve the unbalanced class problem in a number of ways:

1. **Oversampling** where we duplicate data points in the minority class.
2. **Undersampling** where we remove data points from the dominant class.

Both undersampling and oversampling have similar pitfalls. In the case of undersampling, we are removing potentially useful pieces of information for the classifier. Similarly, in the case of oversampling, duplicating a small selection of data points indicates to the model that it has more information about the rare class than it does. Both of these have the potential to cause the model to overfit to an unrealistic sample of the data.

There are a number of methods which attempt to combine oversampling and undersampling, avoiding their individual pitfalls. Having considered a number of approaches, including EasyEnsemble, ADASYN and SMOTE [3], we have decided to make use of the SMOTE method to balance our data set due to it's accessibility and relation to work we have studied perviously. We hope that this will improve the performance of our random forest, in two ways. Firstly, it should ensure that examples of rare classes exist in the training and testing sets for each of our 10 folds. Secondly, it should help us gain more information in the rare classes for the training part.

It is possible that the class ratios in the KDD-99 data set do not represent the ratio of classes in real-world data. For this reason, we assume that all attack types are equally probable. If we don't have a data set that reflects this, the random forest model will have a bias towards DOS attacks that perhaps does not reflect the real-world.

In our project, we undersampled the data by using two method: one is choosing data randomly, the other is using k-means. The first method returns a good result with a high accuracy - 99.4% only using 1279 rows of data, however, the prediction is unfriendly for other attacks events except smurf and neptune especially for nmap, pod, ipsweep and satan. Basically, the more data we used, the more better performance we got. However, it has no improvement as for optimising the performance of the random forest. In that case, we started to think, could k-means bring something different? The key in our undersampling process is that reduce the number of rows for large size of data and k-means could do this by finding k centers for every type of connection. As it took much more efficiency for improve the speed of k nearest neighbors, we were looking forward to seeing a good result in random forest. However, the results are disappointed even if we normalized the data. Since this method is no benefits to the classification of 23 connections, how about the performance of undersampled data in recognize these four types of attacks events?

Synthetic Minority Over-sampling Technique (SMOTE)

Generally, the KDD-99 data set is dominated by DOS attacks and normal connections, however, for other attacks, it doesn't contain enough information which might cause the imperfect performance as for predicting these attacks. In order to improve our model's performance, it is reasonable to try do something to solve this issue by changing the weights for minority sets (may not be discussed here) or re-sampling the data set. For re-sampling data set, it has different ways, such as under-sampling and over-sampling. We've tried under-sampling, so this time, so as to avoid losing too much information, we did oversampling by using SMOTE. In addition, since the performance of smurf and neptune is quite good, for this part, we didn't use them.

SMOTE based on 'interpolation' and k nearest neighbors is a over-sampling algorithm for solving unbalanced issue. So how it works? Suppose we choose $\mathbf{x}$ from the minority, then we randomly choose a sample $\mathbf{x}_i$ from its k nearest neighbors and generate a random number λ between 0 and 1.

$$\mathbf{y}_{new} = \mathbf{x} + \lambda(\mathbf{x}_i - \mathbf{x})$$

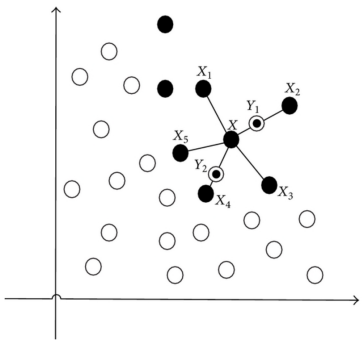

Figure 12: Schematic diagram demonstrating how the SMOTE algorithm synthesises new data points [6]

Intuitively, it creates data points by using their k nearest neighbors' data and this method is also based on the distance. In that case, will it return a better result for distance based model?

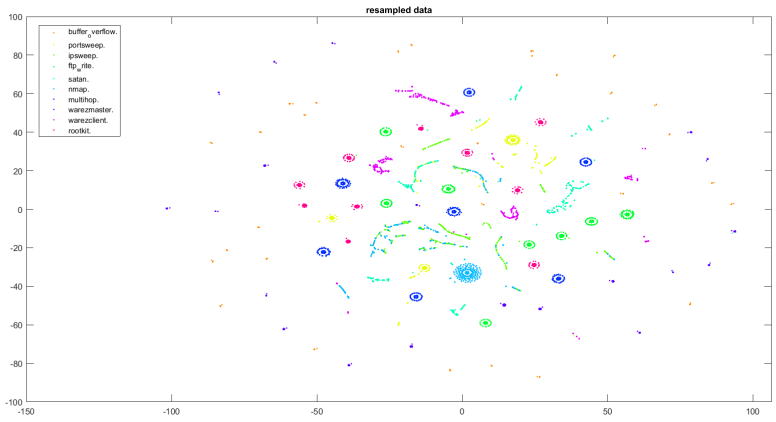

Figure 13: Oversampled data

Compared with original data, it is intuitive that the densities of these data are higher and the characteristics of these attacks are magnified. However, it might be magnified their similar characteristics as well, which is bad for improving our model.

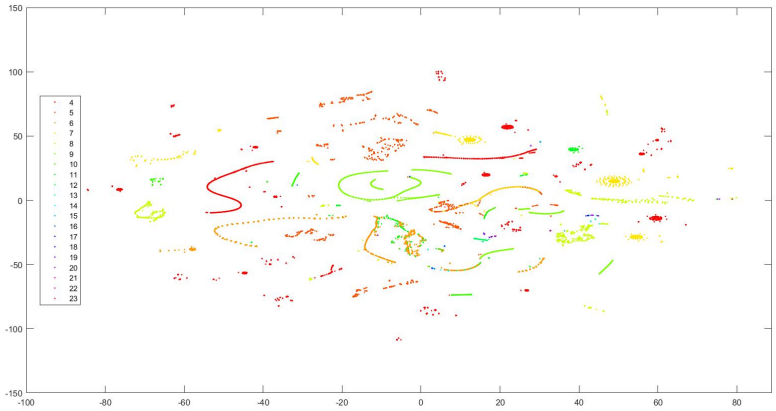

Figure 14: Original data with 20 types of attacks

The performance of oversampling model shows that for some connections, they do deserve more data for training, such as ftp write, multifhop, nmap, rootkit, satan and waremaster which obtained 100% accuracy prediction after oversampling. However, for ipsweep, no matter what oversampling methods like using SMOTE or obtaining data from other data set, it shows no improvement. But the majority of errors just occur to normal, ipsweep and other probe attacks. It makes sense as ipsweep is also a probe attack. Intuitively, as we can see in figure 7, the green one (probe) and the origin one (nmap) they are entangled. The reason might be their behaviours are really close to each other sometimes.

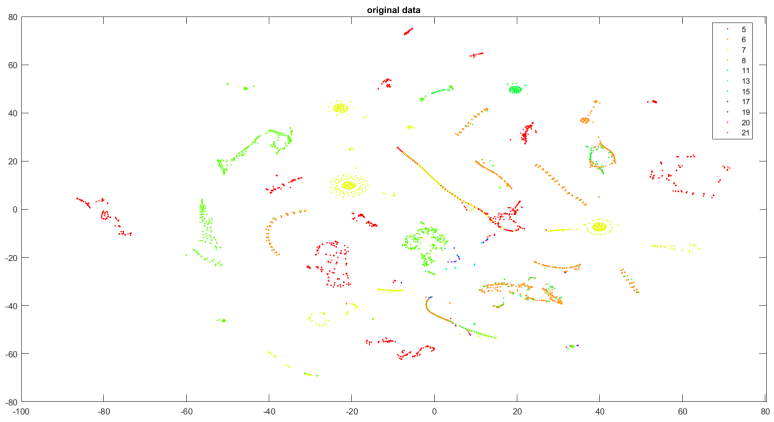

Figure 15: A visualisation of attacks data with low prediction accuracy

As for predicted normal as ipsweep, our assumption is that some normal behaviours are similar to ipsweep behaviours.

Discussion and Conclusions

We have learnt that this data set, from the features (e.g. including packet sizes) to the class sizes is very good for identifying denial-of-service attacks. This can often mask poor performance in other areas. Our initial decision tree shows this quite well; under normal metrics it would be considered very good (it has high accuracy). But it only ever predicts 3 our of the 24 traffic classes!

We attempted to counter this using a number of techniques, including:

- Heavily penalizing misclassifications of rare classes
- Embedded clustering algorithms to provide extra information to the decision tree.
- Oversampling by modelling the rare classes and synthesizing new data points.

However, none of the were as effective as using a random forest.

Given more time we would have liked to investigate embedded more complex and better performing models within the decision trees.

What did we learn about the data?

The dataset is dominated by denial-of-service attacks, and additionally, it has unbalanced data. What we have done is to reduce the unbalance and see if the performance be better.

Further:

- Normal behaviours are likely to regard as attacks sometimes as the normal connections behaviours are wide ranged.
- It's tricky to classify the different types of probe connections, especially ipsweep, even after obtaining more data.
- DOS attacks are the easiest attacks events to detect.

- Even through a few features, we are able to get a good inference for some connections such as land.

Note on out-of-sample performance

All of our models used 10-fold cross validation. This ensures that we never test using data that the model was trained on. This helps to prevent over-fitting, and also decreases the variance in our performance metrics.

References

[1] Bahnsen A. Machine learning algorithms: Introduction to random forests. `http://www.dataversity.net/machine-learning-algorithms-introduction-random-forests/`. Accessed: 2018-11.

[2] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[4] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[5] Ljupčo Todorovski and Sašo Džeroski. Combining multiple models with meta decision trees. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 54–64. Springer, 2000.

[6] Yonghua Xie, Yurong Liu, and Qingqiu Fu. Imbalanced data sets classification based on svm for sand-dust storm warning. *Discrete Dynamics in Nature and Society*, 2015, 2015.

[7] Bernard Zenko, L jupco Todorovski, and Saso Dzeroski. A comparison of stacking with meta decision trees to bagging, boosting, and stacking with other methods. In *icdm*, page 669. IEEE, 2001.