A Comparison of Decision Trees and Random Forest Classifiers on Predicting Malware in Android Apps.

Ross Cradock - 210050274

Introduction

The intent of this study is to build two binary classifiers that can predict whether an Android app is considered to be malware or not based on a dataset [1] of requested app permissions.
The classifications considered will be the Random Forest and Decision Trees algorithms. In both cases the model's accuracy and ability to generalise will be optimised using appropriate hyperparameters.

Exploratory Analysis of the Dataset

- The dataset is available from kaggle.com under the name Android Malware Dataset for Machine Learning.
- It contains 15,036 rows, of which 5,560 are defined as malware apps and 9,476 are considered benign - a ratio of ≈ 1 benign : 0.59 malware.
- The dataset has 215 predictors categorised into groups including OS API calls and OS commands, but these features will be disregarded for this study and only the permissions will be investigated.
- There are 114 binary predictors and one binary target.
- The data used comes from one dataset - DREBIN [2] which contains data from the Android Malgenome project [3]. Samples have been collected over 2010 to 2012 so they are quite dated at this point.
- The apps were downloaded from the Google play store, Russian and Chinese markets along with other sources such as malware forums and security blogs.
- The graph (figure 1) is a subset of 1000 benign and 1000 malicious apps, it plots the total number of permissions requested per app on the x-axis and the number of apps that requested that amount of permissions on the y-axis. Malicious apps are more likely to have around 5-20 permissions requested while benign apps have a more even distribution and show relatively many that request zero permissions.
- The table (figure 2) shows the same subset as before. The ten features shown are those that have the largest difference in the amount of apps that request certain permissions. The USE_CREDENTIALS feature here is used by 170 of the 1000 benign apps but none of the malicious ones.

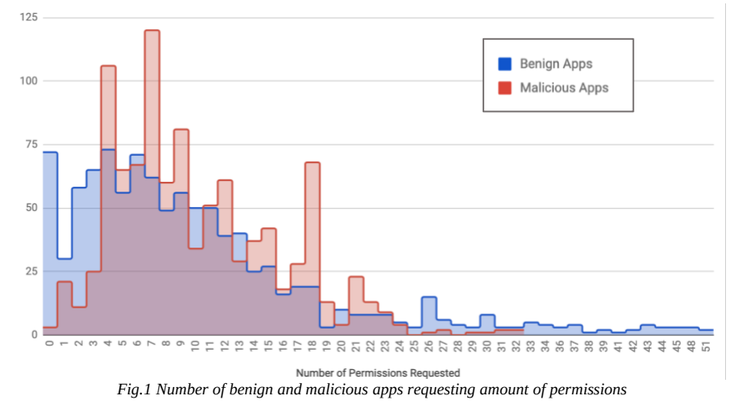

Fig.1 Number of benign and malicious apps requesting amount of permissions

Feature	No. Benign Apps	No. Malicious Apps	All Apps	Difference
SEND_SMS	66	540	606	-474
READ_PHONE_STATE	472	903	1375	-431
GET_ACCOUNTS	427	85	512	342
READ_SMS	75	389	464	-314
RECEIVE_SMS	81	377	458	-296
WRITE_SMS	48	233	281	-185
USE_CREDENTIALS	170	0	170	170
MANAGE_ACCOUNTS	165	2	167	163
RECEIVE_BOOT_COMPLETED	322	485	807	-163
CAMERA	189	38	227	151

Fig. 2 Top ten features showing difference between requested permissions

Model Implementations - Pros and Cons

Decision Tree

The decision tree algorithm is a supervised learning method and is widely used for classification problems. It is a binary tree-structured classifier where each node represents one of the features of the dataset and each of the leaf nodes represents an outcome.

Advantages

- It is human readable and interpretable, so it is possible to see what actions are being taken at each step and what weights are applied to each decision.
- There is less of a requirement for data cleaning operations like standardisation or normalisation.
- Decision trees will implicitly perform feature selection as part of their process which can help with datasets that have a large number of features [4].

Disadvantages

- Decision tress can often have problems of overfitting which has a very small error rate on the training data but does not generalise predictions well.
- An increase in class labels will increase the complexity. In this circumstance this shouldn't be an issue with only two output labels.
- Decision trees can become biased if there is one dominant class, datasets may need to be balanced to avoid this.

Random Forest

Random forest is a supervised machine learning algorithm which aggregates the output of multiple decision trees into a single output model. Decision trees are built from randomly selected samples and features using a process of gradient boosting to achieve the best accuracy or bagging which relies on majority voting (in classification) to produce a "forest" of decision trees.

Advantages

- Compared to decision trees, random forest will exhibit better generalisation of a dataset and will avoid issues of overfitting by aggregating across many decisions of many trees [5].
- Can be used for both regression and classification problems.
- The decision trees can be created in parallel because they are considered to be independent of one another, reducing training time.

Disadvantages

- The larger number of generated decision trees mean that the training can take longer and be more computationally expensive.
- The model and process through which an outcome is reached is less human readable because of the increased complexity.

Hypothesis

I assert that both the decision trees and random forest algorithms will be able to discern which Android apps are malware and which are not, to a degree of certainty that is well outside random chance. The large number of features in this dataset mean that decision tress will have a high degree of variance while random forests will suffer from time consuming processing. I assert that random forests will be able to outperform decision trees due to bagging of the larger number of features and ability to better generalise from training data.

Training and Evaluation Selection

Training

1. The full dataset is split up so that there is 70% (10,525 rows) used for training while 30% (4,511 rows) is unseen and used for testing against a chosen model.
2. K-fold cross validation will be applied to the training data. Partitioning of between 2 and 20 folds will be tested and a number will be selected for error rate and time consumption.
3. Misclassification error rates and confusion matrices will be used to understand the models and optimise selected hyperparameters to achieve best results.
4. The models will be then used to predict against the unseen test dataset.

Evaluation

1. Accuracies will be based on the misclassification error rates using K-fold cross validation.
2. The accuracies of the decision tree and random forest models will be compared to each other.

Parameters and Experimental Results

Decision tree

For the decision tree all the following hyperparameters were tested with a default Kfold cross validation of 5 initially to get a preliminary understanding of which ones are most impactful on the accuracy:
Maximum number of splits, minimum leaf size and split criterion (between Gini diversity index and maximum deviance reduction). The optimize hyperparameter argument was used which ran 30 iterations.
Of the top 10 best estimates, all of them had the same objective value and they all had a minimum leaf size of 1 and a maximum number of splits greater than 1200. This indicates that tuning the hyperparameters further would not improve the model. The accuracy was 94.4%.
The number of Kfold cross validations to be used seemed to have an effect on the accuracy more so than other the above hyperparameters, so a test was created to try and find the most reliable and accurate Kfold number to use.
10 iterations of the test were conducted looking at the training a decision tree with Kfold numbers between 2 and 20 for both Gini diversity index and deviance reduction. A polynomial fit of the average of all 10 iterations is plotted against accuracy and it can be seen that Kfold numbers between 10 and 16 have about the same accuracy. 12 Kfolds was selected to gain the benefit of being more time efficient.
Training was then done on the data and an accuracy of 94.6% was achieved. This was the same result seen over the top 3 best results. The training was done with all hyperparameters again optimized since the time required to run 30 iterations was feasible.

Random Forest

In the case of a random forest, both bagging and boosting methods can be used.
The hyperparameters for bagging is the number of bagged trees. This was set initially at 50 and it can be seen in figure 3 that the error rate reduces drastically in the first 10 bagged trees but only has very slight improvements from around 15 onwards. As such the bagged model was saved, having achieved an out-of-bag error rate of 4.49%.
There are more hyperparameters for the boosted random forest including the number of learning cycles used, the learning rate and the method of boosting between these options: GentleBoost, LogitBoost, AdaBoostM1 and RUSBoost.
An initial hyperparameter optimization was run 30 times with default values to take a preliminary estimate at the most impactful hyperparameters. It was clear to see from this that higher numbers of cycles (greater than 200) and lower learning rates (less than 0.05) had high accuracy. So setting these 2 parameters at values of 200 and 0.001 respectively, a test was conducted on methods only. 5 tests of each boosting method were completed to determine which was the most effective. Only GentleBoost managed to get accuracies as high as 95% so only this was used.
Having now controlled for the method, a final test of hyperparameter optimisation was conducted with learning rate and number of cycles. Output from this optimisation can be seen in figure 4.

Unseen Test Data

Once the models were trained, the most accurate were saved and used against the unseen test data to determine how well the models could generalise across this dataset and not just within the training sets.
The models and test data were loaded into a script and accuracies were determined by the misclassification error rate of each. **Decision tree - 95.21%, Random Forest Boosted - 95.99% and Random Forest Bagged - 96.07%**

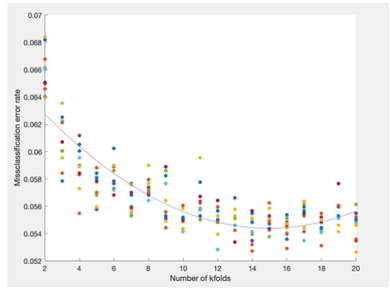

Fig. 3 DT - GDI Kfolds

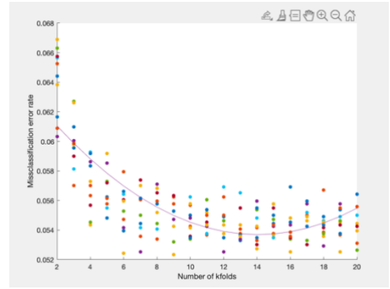

Fig. 2 DT- Deviance Kfolds

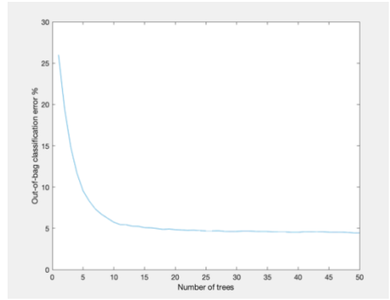

Fig. 3 RF - Bagged error rate

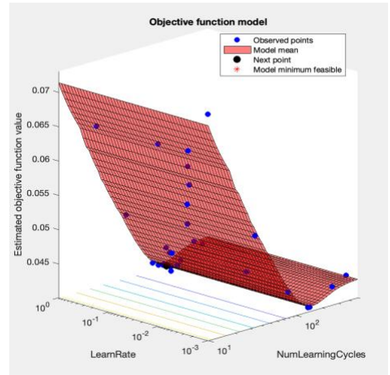

Fig. 4 RF - Error rate for learning cycles and learning rate

Analysis and Evaluation of Results

From the outset of testing both algorithms against this dataset, a very high level of accuracy can be seen, this appears to be due to the very deterministic nature of the binary input data being only one of two distinct options each time and as such reducing the number of variables to be considered as opposed to scalar values.
It can be seen that in both decision tree and random forest algorithms that there is very little improvement over initial results before the optimisation and also between the algorithms. The nature of the binary input data appears to have had a profound effect on this behaviour. Decision trees, and by extension random forest, work very well with fine tuning scalar values at each node to better produce more generalised and accurate models. In the case for each node in this dataset the decisions were effectively yes or no. Looking more closely at the decision tree, each node has a value criterion for each feature of < 0.5 and >= 0.5 and these values do not change with further training or optimisation.
It appears that the optimisation that occurs for the random forest will involve changing the order or location in which the nodes are processed inside the tree. This does show some improvement but only marginally and not to an extent that would indicate that this improvement couldn't be seen as no more than a statistical variation that occurs when many trees are created and the best one is selected, instead of one trial and one selection. To illustrate we can take the example of the first trained decision tree which had a Kfold cross-validation accuracy of 94.4% and compare this to the most optimised random forest model with an accuracy of 96%. An increase in accuracy of only 1.6%.
The dataset used has a very large number of features but was still very easy to work with for training. None of the models took a very long time, only the random forest boosting hyperparameter optimisation took more than 5 minutes for 30 iterations. This allowed for easy experimentation especially with the decision tree. I was able to run through 360 trainings very quickly while varying Kfolds and split criterion. Although when considering the time-cost benefit of improving a model for this kind of dataset, the gains would, at least from this experiment, most likely be not worth the effort. Instead improvements to the dataset itself would be more fruitful.
The dataset at this point is very old and malware developers along with average users will have become more wary of apps that request a lot of permissions. Another issue with using this dataset is that the ratio of malware to benign apps is far from what would be expected in the real world, inducing a sample bias. A normal Android smart phone would not expect to see 37% of the apps downloaded being malware - this imbalance would inherently increase the rate of false positive alerts for malware.

Lessons Learned and Future Work

Binary input data is highly deterministic and in most cases a decision tree or a random forest cannot be improved upon to great effect.
Binary data is very quick to train an algorithm so even very large datasets can be used efficiently.
Random forest is highly adaptable having a broad set of use-cases with both classification and regression problems as well as having a large number of parameters that can be optimised for different scenarios.
The dataset is nearly 10 years old at this point and a lot has changed with the Android operating system since then so more up to date data would need to be gathered and possibly with new features to track the changing landscape of Android malware.
Explore other algorithms that might be able to better deal with binary data such as a support vector machine [6].

References

[1] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," IEEE Transactions on Cybernetics, vol. 49, no . 2, pp. 453–466, Feb. 2019, doi: 10.1109/tcyb.2017.2777960.
[2] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," Feb. 2014. Accessed: Dec. 14, 2021. [Online]. Available: https://prosec.mlsec.org/docs/2014-ndss.pdf.
[3] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symposium on Security and Privacy, 2012, pp. 95-109, doi: 10.1109/SP.2012.16
[4] K. Grabczewski and N. Jankowski, "Feature selection with decision tree criterion," Fifth International Conference on Hybrid Intelligent Systems (HIS'05), 2005, pp. 6 pp.-, doi: 10.1109/ICHIS.2005.43.
[5] L. Breiman, "Breiman randomforests," 1999. Available: http://machinelearning202.pbworks.com/w/file/fetch/60606349/breiman_randomforests.pdf
[6] Stanevski, N. and Tsvetkov, D., 2005. Using support vector machine as a binary classifier. In International Conference on Computer Systems and Technologies–CompSys Tech