Machine Learning for Software Development Project - Malware Classifier

Ayushi Churiwala 7010542

Introduction:

I intend to implement a simple malware classifier given the metadata extracted from Android applications and App Repositories for the given train and test set. The training set consists of metadata collected from 7922 Benign and 14117 Malware apps and the testing set consists of metadata collected from 4405, both Benign and Malware apps with 181 features each in the metadata. I will compare results of both machine learning algorithms and a simple deep learning neural network to evaluate the classification task.

Data Pre-processing:

The training set has 181 features in the metadata with the target label as “Malware” which is 0 if the app is benign and 1 if it is malicious. Apart from this, the data set has 5 qualitative features namely, “App”, “Package”, “Category”, “Description” and “Realated Apps”. Rest 175 features are quantitative such that 173 of them are fields containing Android runtime permissions with binary values. The remaining two columns specify Dangerous and Safe permissions count. Further analysis of data led to some findings and following ways were implemented to address them:

1. The dataset has the class imbalance problem such that there are more malicious apps(14117) as compared to benign apps(7922). This can cause the machine learning classifier to be more biased towards the majority class, causing bad classification of the minority class. To avoid this downsampling and upsampling techniques were performed on the training set. It was observed that downsampling gave poorer results(maybe the sample is not enough for learning the classification task) for our models and hence I chose upsampling by randomly adding apps from the training set with the minority class. I have also shuffled the data set as the training set has benign and malicious apps one after the other.

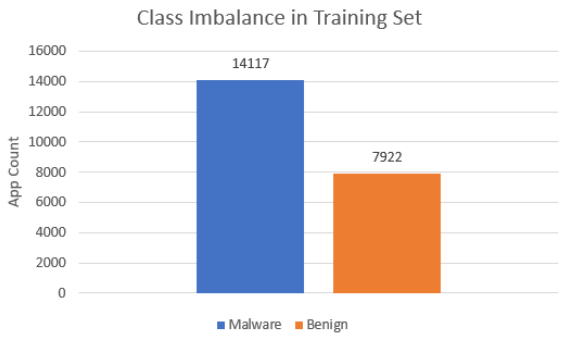

Figure 1: Class Imbalance

2. Several rows contain missing values for “Related apps”, “Dangerous permissions count”, “Description”, “App” and “System tools: set wallpaper (S)”. To handle this, I have filled the

missing places with zeroes or empty string values for “Related apps”, “Dangerous permissions count” and “Description”. I have removed rows with no App name as application must have a name and this row could have been mistakenly added.

	NaN %
Related apps	2.57
Dangerous permissions count	0.76
Description	0.09
App	0.00
System tools : set wallpaper (S)	0.00

Table 1: NaN counts

3. Further upon analyzing the qualitative features it was observed that several applications have the same app name, package names (not necessarily the same app) and related apps. Surprisingly, some apps have the exact same descriptions as well. Hence, these features could be used for classification but since some have both classifications I decided to not include “App” and “Package” as features to determine the app behaviour. I have removed rows with applications which have duplicate rows with both the same and different classification values.

	App	Package	Category	Description	Related apps
count	22038	22039	22039	22020	21473
unique	17904	18299	30	18170	18294
top	Tic Tac Toe	com.apostek.SlotMachine	Entertainment	Phrasebook and Translator contains all the ess...	{com.openkava.spinpic}
freq	39	8	1962	27	33

Table 2: Shows total counts, unique counts, most occurring values and their frequencies.

App	Package	Category	Description	Malware
Tic Tac Toe	mk.g6.ikselent	Brain & Puzzle	Tic Tac Toe provides few interestin	0
Tic Tac Toe	com.mobiloids.tictacktoe	Brain & Puzzle	"Tic Tac Toe"<p>The be	0
Tic Tac Toe	mk.g6.ikselent	Brain & Puzzle	Tic Tac Toe provides few interestin	0
Tic Tac Toe	tm.app.ticTacToe	Casual	Tic Tac Toe game for Android. Play	1
Tic Tac Toe	mk.g6.ikselent	Brain & Puzzle	Tic Tac Toe provides few interestin	1

Figure 2: Shows same app features with different classification labels.

4. Feature “Category” was encoded using a Label Encoder and the encoded values were used for the classification task.
5. “Related apps” was also encoded using a Label Encoder to see if apps having the same related apps are always getting classified similarly or not. One Hot Encoding was not used as it would create a large sparse tensor. However, I believe other ways (not explored here) of encoding this feature could be helpful for the classification task.
6. “Description” has also been excluded. Although, creating a dictionary with malware or security related words and using it to create embeddings could improve classifying malware if the apps have sensible descriptions.

Feature Selection and Transformation:

We can see the variation in classification against some features visually:

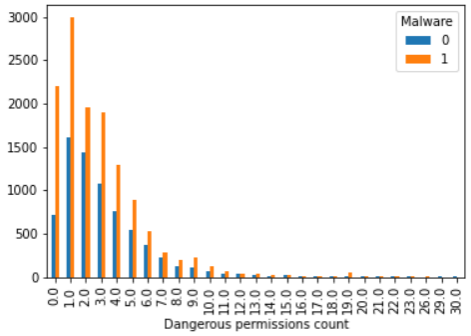

Figure 3: Malware vs number of Dangerous permissions

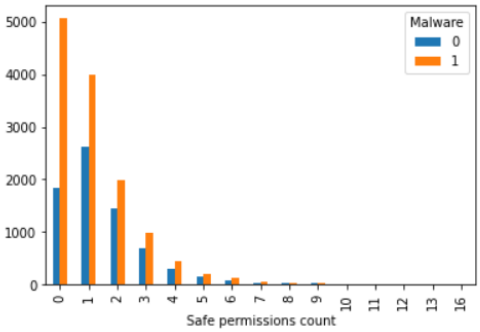

Figure 4: Malware vs number of safe permissions

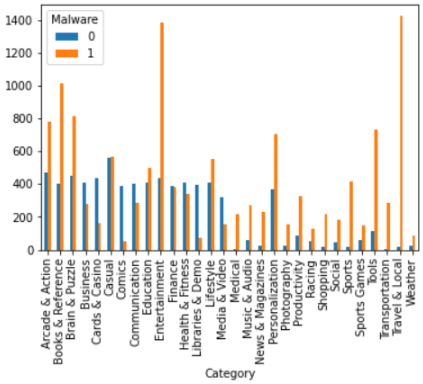

Figure 5: Malware vs Category

Feature selection is used to reduce the number of input features to those that are supposed to be most useful for the model in order to predict the target variable. I experimented with correlation based, tree based and a univariate feature selection(χ^2) method. The correlations did not yield strong values and amongst the others, both feature selection methods gave similar results. I chose the tree-based selection method.

Further, the number of features to be used was experimented with such that 20 top features were finally chosen. Related apps and Category are amongst the top features. It was also seen that for the same related apps the malware classification was the same. Features have been normalized using the standard scaler present in sklearn library.

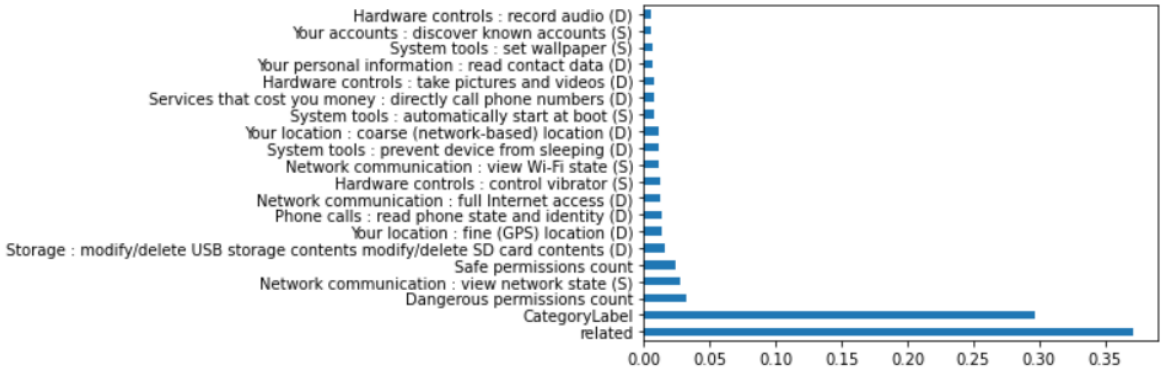

Figure 6: Shows Top-20 features selected by ExtraTreesClassifier with their feature importance values

Neural Network Architecture:

I used a simple deep learning based neural network model with linear layers. Below is the architecture used for classification:

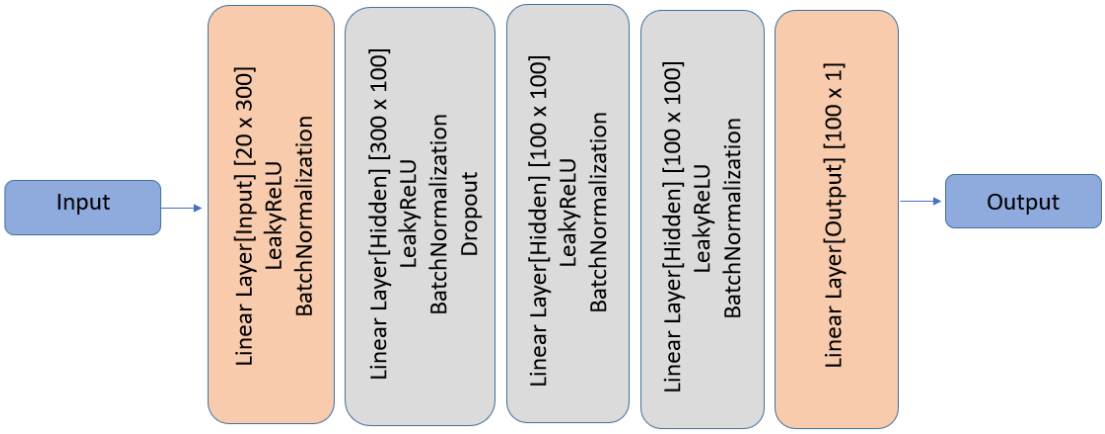

Figure 7: Neural Network architecture

I have used 3 hidden layers with Batch Normalization layers after the activation functions. Using Batch Normalization before the activation function did not improve the model. It improves the precision but the recall falls drastically. For activation function LeakyReLU was used. I also tried ReLU and SeLU. I have used a dropout of 0.1 for the first hidden layer. Binary cross-entropy loss has been used with Adam optimizer. SGD did not perform as well as Adam. The learning rate with best results was for 0.001. I tried adding an additional L1 loss to the BCE Loss but it did not help much. I have implemented early stopping, batch normalization and dropout to avoid overfitting of the training data.

Evaluation:

The model has been run for 100 epochs with a batch size of 64. Below are the training loss and accuracies and the test accuracies for epochs in the range of 0 to 100 at intervals of 10 epochs. I choose the model with 20 epochs as for more epochs even though the training accuracies and loss improve but they are not very significant.

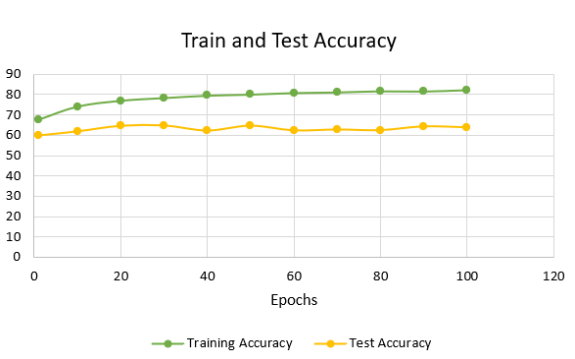

Figure 8: Train and test accuracies over 100 epochs

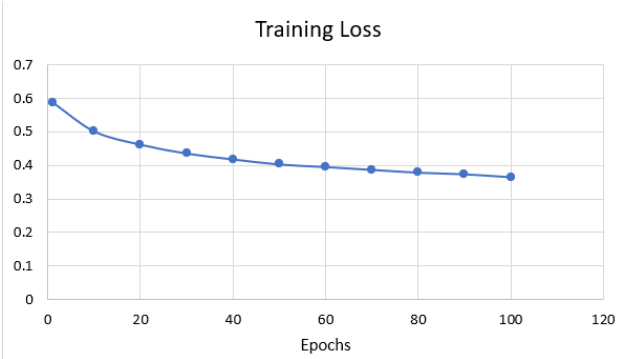

Figure 9: Loss over 100 epochs

	precision	recall	f1-score	support
0	0.34	0.60	0.44	989
1	0.85	0.66	0.74	3315
accuracy			0.65	4304
macro avg	0.60	0.63	0.59	4304
weighted avg	0.73	0.65	0.67	4304

Figure 10: Evaluation scores

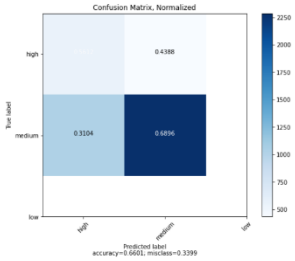

Figure 11: Confusion Matrix

Other evaluations:

I have used several ML models from sklearn library for comparing the neural network. I started with a null model and used forward selection for hyperparameter tuning. The data pre-processing steps are the same as used in the neural network. Using a backward selection was extremely time consuming. I have chosen best model parameters from the incremental models and compared them with the neural network outputs. The comparison should not be made based on accuracy scores alone as the model can easily learn to overfit and give around 77% training accuracy as there is class imbalance. Hence, both precision and recall for the two classifications- Malicious or Benign should be considered while selecting a model.

Model Name	Classification Label	Precision	Recall	F1-score	Test Accuracy
Naïve Bayes	0	0.69	0.3	0.42	55.32
	1	0.51	0.85	0.64	
Random Forest Classifier	0	**0.77**	0.32	**0.45**	56.46
	1	0.5	**0.88**	0.64	
Logistic Regression	0	0.71	0.31	0.43	56.66
	1	0.52	0.86	0.65	
Gradient Boosting Classifier	0	0.61	0.32	0.42	60.92
	1	0.61	0.84	0.71	
SVM	0	0.66	0.33	0.44	61.31
	1	0.6	0.86	0.7	
KNN	0	0.64	0.31	0.42	58.8
	1	0.57	0.84	0.68	
Neural Network	0	0.34	**0.6**	0.44	**64.59**
	1	**0.85**	0.66	**0.74**	

Table 3: Evaluation metrics for different models

I tried two other variations of the neural network proposed. One with the same structure but trained and tested on only word embeddings of “Description” feature and the other one trained on both the top-20 features and the word embeddings for “Description”. Here, I use spacy for tokenization and glove embeddings from “glove-twitter-50”. These two models are not as good as the simple Neural Network model. I believe however, description seems to be an important feature and building a vocab of security related words and using some other model architecture can be explored.

Model Name	Classification Label	Precision	Recall	F1-score	Test Accuracy
Neural Network	0	0.34	**0.6**	0.44	**64.59**
	1	**0.85**	0.66	**0.74**	
Neural Network with Word Embeddings of Description only	0	0.32	**0.63**	0.42	**60.03**
	1	**0.84**	0.59	**0.7**	
Neural Network with top-20 features and Word Embedding	0	0.23	**1**	0.38	**23.9**
	1	**0.98**	0.01	**0.02**	

Table 4: Evaluation metrics for different neural network models

Conclusion:

It seems like all models perform fairly well with the neural network giving highest test performance. For malware classification focusing on the False Negatives is more important I believe and it should be given more importance. Also, as the dataset was imbalanced, I would consider choosing a model that gives higher precision and recall for benign classifications. Based on these points the Random Forest Classifier and the simple Neural Network perform well and I choose the Neural Network model here. However, introducing word embedding for Descriptions based on a security vocabulary can be something to work on as future improvements.

Appendix:

Naïve Bayes:

BernoulliNB(fit_prior = 'True')

Random Forest Classifier:

RandomForestClassifier(random_state=50, n_estimators=50, max_depth=3, max_leaf_nodes=3, min_samples_leaf = 3)

Logistic Regression:

LogisticRegression(max_iter= 100, solver = 'sag', C= 0.08858667904100823, penalty='l2', random_state=10)

Gradient Boosting Classifier:

GradientBoostingClassifier(learning_rate=0.1, subsample=0.9, max_features=13, min_samples_leaf=30, min_samples_split=200,max_depth=29, n_estimators=130, random_state=10)

SVM:

SVC(degree=1, C=100, kernel='rbf', random_state=10)

KNN:

KNeighborsClassifier(metric= 'manhattan', weights='distance',n_neighbors=5)

NN:

BinaryClassification(
  (layer_1): Linear(in_features=20, out_features=300, bias=True)
  (layer_2): Linear(in_features=300, out_features=100, bias=True)
  (layer_3): Linear(in_features=100, out_features=100, bias=True)
  (layer_4): Linear(in_features=100, out_features=100, bias=True)
  (layer_out): Linear(in_features=100, out_features=1, bias=True)
  (relu): LeakyReLU(negative_slope=0.01)
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True))