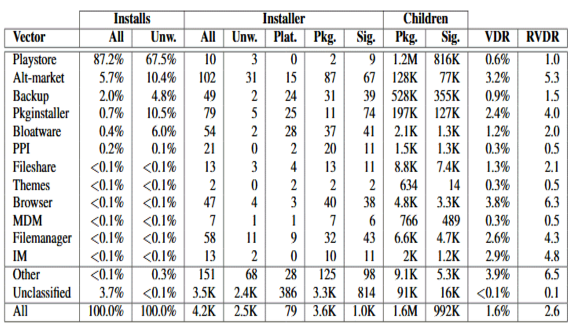
Android Malware Detection using App permissions

Abhimanyu Gupta 2019226
Shubham Lohan 2019275
Pankaj Kumar 2019262
Vasu Kashyap 2019343

1. Motivation

] With the increasing boom in the android market, there is a constant increase of apps with malicious activities. According to ZDNet, 10-24% of apps over the Play store could be malicious applications. On the surface, these apps look like any other standard app, but they exploit the user system in various harmful ways. The current methods to detect malwares are both resource heavy and exhaustive, yet fail to compete with the pace of new malwares.

TABLE VII: Summary of app distribution.



	Installs		Installer					Children			
Vector	All	Unw.	All	Unw.	Plat.	Pkg.	Sig.	Pkg.	Sig.	VDR	RVDR
Playstore	87.2%	67.5%	10	3	0	2	9	1.2M	816K	0.6%	1.0
Alt-market	5.7%	10.4%	102	31	15	87	67	128K	77K	3.2%	5.3
Backup	2.0%	4.8%	49	2	24	31	39	528K	355K	0.9%	1.5
Pkginstaller	0.7%	10.5%	79	5	25	11	74	197K	127K	2.4%	4.0
Bloatware	0.4%	6.0%	54	2	28	37	41	2.1K	1.3K	1.2%	2.0
PPI	0.2%	0.1%	21	0	2	20	11	1.5K	1.3K	0.3%	0.5
Fileshare	<0.1%	<0.1%	13	3	4	13	11	8.8K	7.4K	1.3%	2.1
Themes	<0.1%	<0.1%	2	0	2	2	2	634	14	0.3%	0.5
Browser	<0.1%	<0.1%	47	4	3	40	38	4.8K	3.3K	3.8%	6.3
MDM	<0.1%	<0.1%	7	1	1	7	6	766	489	0.3%	0.5
Filemanager	<0.1%	<0.1%	58	11	9	32	43	6.6K	4.7K	2.6%	4.3
IM	<0.1%	<0.1%	13	2	0	10	11	2K	1.2K	2.9%	4.8
Other	<0.1%	0.3%	151	68	28	125	98	9.1K	5.3K	3.9%	6.5
Unclassified	3.7%	<0.1%	3.5K	2.4K	386	3.3K	814	91K	16K	<0.1%	0.1
All	100.0%	100.0%	4.2K	2.5K	79	3.6K	1.0K	1.6M	992K	1.6%	2.6

What can help us to overcome these challenges ?

- A strategy that can assess and analyse the data of confirmed Malicious Applications.
- A model that can accurately predict the Malicious Application based on the permissions.
- Proposing a machine learning malware detection model that relies on metadata information available publicly. evaluating such model and assessing its potential as a first-stage malware filter to detect Android malware

2. Introduction

Despite the increasing malwares, there is not yet an effective and robust method to detect malware applications. With the increasing applicability of Machine Learning in various domains, we believe the issue of detecting Malware can be solved using Machine Learning techniques. Our project aims at a detailed and systematic study of malware detection using machine learning techniques, and further creating an efficient ML model which could classify the apps into benign(0) and malware(1) based on the requested app permissions. This study Proposes

- Examining and Evaluating Android metadata and Permissions as Malware Predictors
- Proposing a machine learning malware detection strategy that relies on publicly available metadata information.
- Analysing such a model and determining its utility as a first-stage malware filter for Android malware detection

3. Literature Review

Paper 1:

In the paper titled Dynamic Permissions based Android Malware Detection using Machine Learning Techniques, the authors talk about various Machine Learning techniques which could be used for Malware Detection in Android apps using permissions. The study has been conducted on an appreciable sample size of 11,000 apps with close to an equal number of benign and malicious apps. All the used terminologies are defined properly and well explained. The used methodology has been also well described which empowers others to perform the study themselves. Adequate use of graphs and tables to present vital facts/figures whenever necessary, makes it easier to follow through the steps of study. Though the author demonstrates the performances of Machine Learning techniques, they don’t compare between the feasibility, practicality and performance of pre-existing classical techniques and new ML techniques. Further, the study also fails to deliver why one ML technique outperforms the other. Another criticism against the paper is its lack of reasoning on why the app permissions stand out to be such a good measure for classifying benign and malicious applications.

Paper 2:

In the paper titled Machine Learning for Android Malware Detection Using Permission and API Calls authors start by laying out the different techniques used in circulation of Malware and current Malware detection techniques to fight against them. They then point out why such solutions like traditional Signature based approaches are not good enough. Moving further into the premise an emphasis on why application permission acts as a great tool for Malware Detection is made by presenting Android Application structure in much detail. Though the methodology explained is decent, exact details of technologies used are not provided, which raises an eye on the credibility. The paper provides 3 Machine Learning methods using which experiments were done over some 2510 apps containing 1260 malwares. The paper lacks a proper description of used techniques and also fails to analyse their performances. Future prospects and the ways the study can be extended are not talked upon.

4. Dataset Description

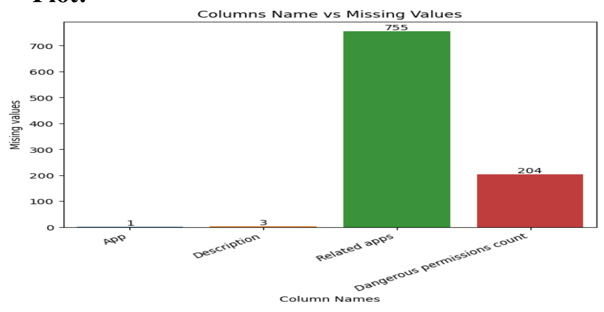
Details:

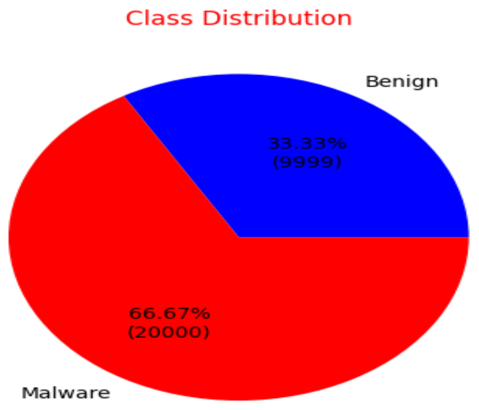
- Dataset has been taken from kaggle

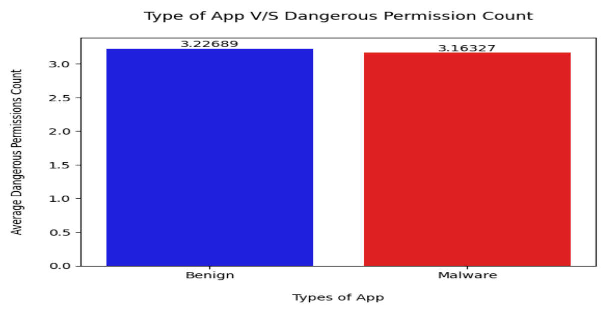
  Data contains the details of the permission of almost 30k app
- There are 183 features in the dataset like Dangerous Permissions Count, Default : Access DRM content, Default : Move application resource, etc.
- There is one target class(binary- 0/1) named - 'Class', indicating Benign(0) and Malware(1) applications.
- There are 29,999 records with 20,000 malwares and 9,999 benign apps.

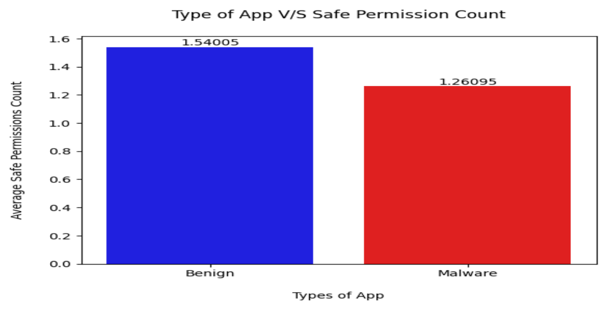
Preprocessing, Visualization and Analysis: Data is read from a csv file into a dataframe for easy use. Required attributes are filtered out from the dataset. Several plots are built to better understand/analyse the data. Date is checked for null/missing values and are therefore replaced by the mean of the column. Data is then analysed on the basis of the distribution of Malware and Benign applications in various settings and several plots were made to visualise the results. Matplotlib and Seaborn are used for plotting and visualization. Removed all other columns having information other than permissions. Mapped app names to index to easily access the information.

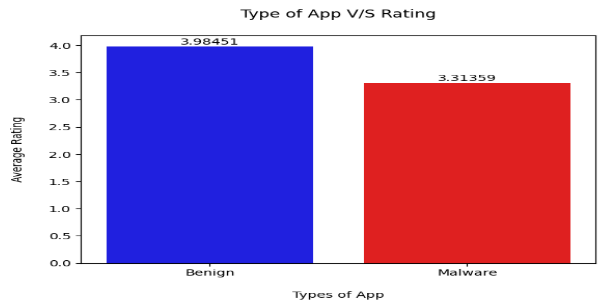
Plot:

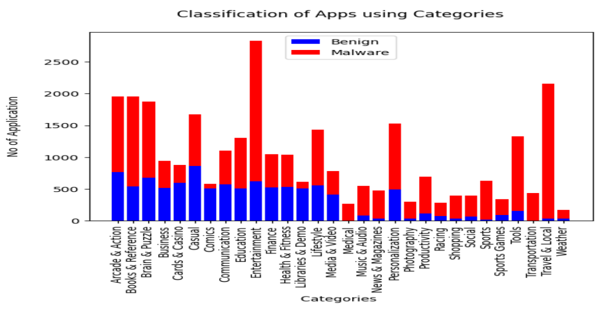












5. Methodology

After Preprocessing the data, data is split into testing and training sets on a 8:2 ratio. We have done the Under and Across Sampling over the Dataset, however the outcome don't appears promising at the end. Following the sampling, we used different classifiers, including logistic regression, decision trees, and Naive Bayes. However, the outcomes are unsatisfactory.

However, after inspecting the Dataset, we see that there are several multivariate data tables, thus we must apply PCA to each Dataset. We plotted the Variance Percentage after using the PCA. As a result, we chose to use the Inverse transform. It is now up to us to apply the classifiers to the provided dataset. First, we used Random Forest, which resulted in a considerable improvement in the supplied accuracies. Following that, we used the Boosting approach to increase their prediction accuracy. We used the boosting strategy on an unsampled dataset and on one after selecting Reliable features, and the results show that the model is improving. Finally, we used SVM and MLP to the final dataset and obtained our best results. When we compare the results obtained after feature selection and boosting, we can see that we have progressed and obtained the final accuracy.

6. Results and Analysis

Model Trained on the UnSampled Data

Decision Tree (Criterion=Gini ,max_depth=10,max_leaf_nodes=10)	
Precision	0.7306158617634028
Accuracy	0.6791666666666667
Recall	0.8230596456201648
Roc_Auc	0.6064620857303031

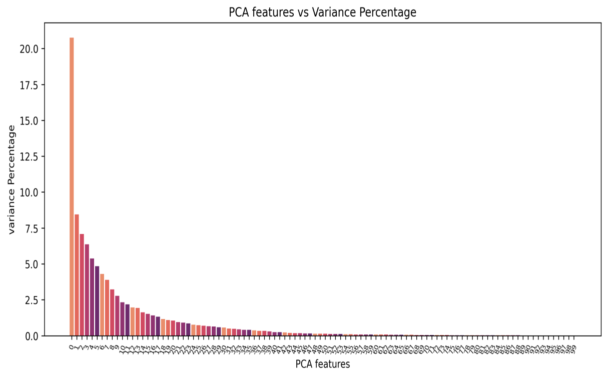
Logistic Regression (Default)	
Precision	0.6682820855614974
Accuracy	0.6678333333333333
Recall	0.9980034938857
Roc_Auc	0.5010087715289011

GaussianNB (Default)	
Precision	0.9617117117117117
Accuracy	0.5371666666666667
Recall	0.3196905415522835
Roc_Auc	0.6470504890400655

Overall

Classifier Algorithm	**Optimal parameters**	**Precision**	**Accuracy**	**Recall**	**ROC_AUC**
Logistic Regression	test_size=0.2, random_state=42	0.66828208 55614974	0.66783333 33333333	0.99800349 38857	0.50100877 15289011
Gaussian NB	Default	0.96171171 17117117	0.53716666 66666667	0.31969054 15522835	0.64705048 90400655
Decision Tree	Criterion=Gini, max_depth=10, max_leaf_nodes=10	0.73061586 17634028	0.67916666 66666667	0.82305964 56201648	0.60646208 57303031

As we seen in the Tabulation that, Accuracy follows the order as follow:
Decision Tree > Logistic Regression > Gaussian Naive Bayes
This were the results before we had done sampling and under-sampling.



Logistic	Optimal Parameter	Accuracy	Recall	ROC
SVM	default	Training Accuracy 0.85 Test Accuracy 0.85	0.94	0.80
Random Forest	n_estimators=200, n_jobs = -1	Training Accuracy 0.87 Test Accuracy 0.86	0.93	0.81
MLP	random_state = 42, max_iter = 300	Training Accuracy 0.85 Test Accuracy 0.85	0.95	0.80

By looking at the result we can say that Random Forest perform best among all the classifiers with Accuracy of 86%. As we seen in the Tabulation that, Accuracy follows the order as follow: **Random Forest > MLP > SVM**

7. Conclusion

- Learning-

  Different ways to visualize the data for better understanding of features. Machine Learning models like Logistic Regression, Naive Bayes and Decision Tree to model the problem. How to use platforms like Kaggle and Google Colab. How to work and collaborate in teams.

- Work Left- We conducted study, but there is always opportunity for some improvement. We hope that our project will be useful in future research.

- Member Contribution
  - Abhimanyu - Preprocessing, Report Writing, Result Analysis, Training Model, Data Visualization,Literature Review,Data Preprocessing, Feature Selection

    Shubham - Data Preprocessing, Data Visualisation, Model Training, Model Testing, Result Analysis,Report Writing, Model Tuning
  - Pankaj - Report Writing, Exploratory Data Analysis, Model Selection, Model Testing, Result Analysis, Model Tuning
  - Vasu - Data Preprocessing, Data Visualisation, Exploratory Data Analysis,,Feature Selection Report Writing

References

[1] Dynamic Permissions based Android Malware Detection using Machine Learning Techniques

[2] Machine Learning for Android Malware Detection Using Permission and API Calls

[3] Android Permission Dataset