Detection of Malware Based on Android App Permissions

UNSUPERVISED ALGORITHMS IN MACHINE LEARNING

https://github.com/slilly4/UnsupervisedLearning

Problem Statement

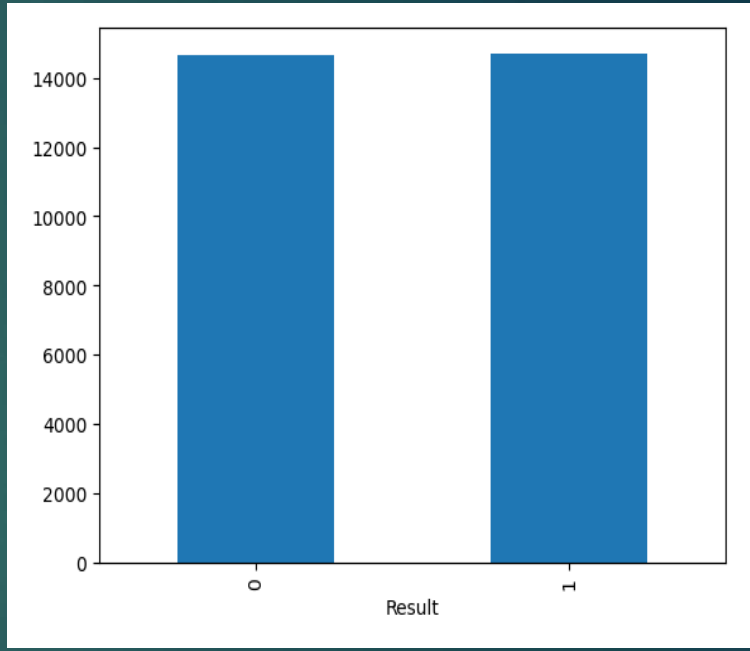
- This project endeavors to apply machine learning algorithms to a dataset of Android app security permission settings to determine whether a given app is malware or not. Since providing labels for supervised learning techniques can be time consuming and difficult, I would like to determine whether unsupervised methods can provide comparable results to the supervised ones. If so, unsupervised learning methods could greatly increase the ease by which apps are classified and flagged.

Data

- The data for this study was collected from 29,000 Android apps between 2010-2019.
- This data was first published in 2021 in conjunction with a research paper entitled, *NATICUSdroid: A malware detection framework for Android using native and custom permissions*. It is freely available through the UCI Machine Learning Repository.
- Citation:
  - *Mathur, Akshay. (2022). NATICUSdroid (Android Permissions) Dataset. UCI Machine Learning Repository. https://doi.org/10.24432/C5FS64.*

Data Attributes

- 29,332 objects with 87 features
- The data is does suggest collinearity. Several features have a VIF over 10. For training with Logistic Regression, features with the highest VIF will be iteratively removed until all have a value below 10.
- The data is ordered, so shuffling will take place
- No data needs to be imputed
- All data is binary, and no scaling is needed



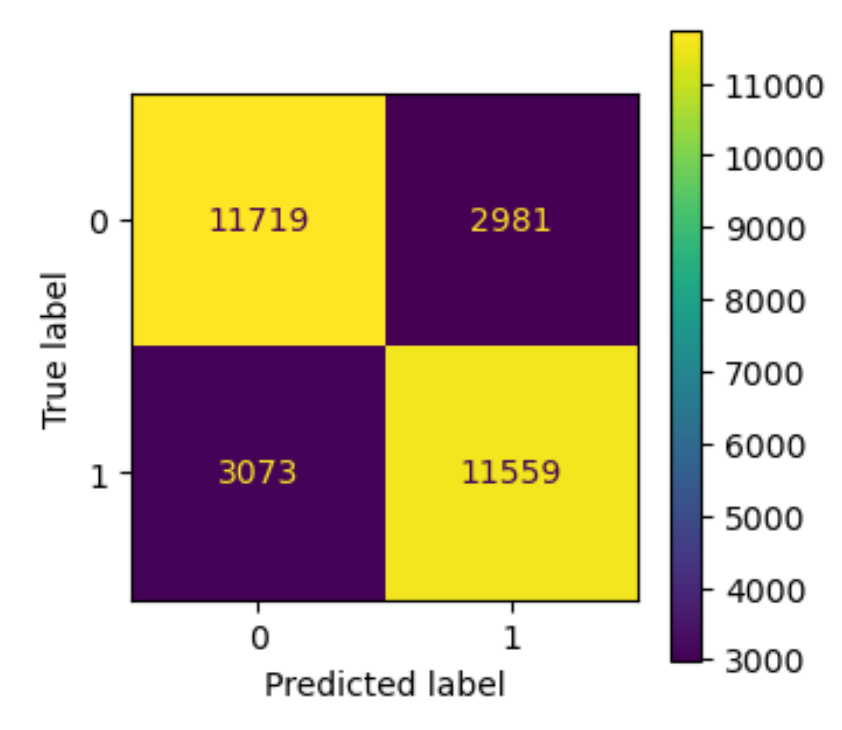
Machine Learning Approach

- Unsupervised Methods: Non-Negative Matrix Factorization and K-Means algorithms
- Supervised Methods: Singular Value Decomposition, Logistic Regression, K-Nearest Neighbors, Random Forest, Support Vector Machines, and Gradient Boosting algorithms
- These will be compared based on accuracy, precision, recall, f-score, confusion matrix and AUC.
- Will use grid search where applicable to establish optimal hyper-parameters
- Models susceptible to collinearity will be trained again on data excluding select correlated features

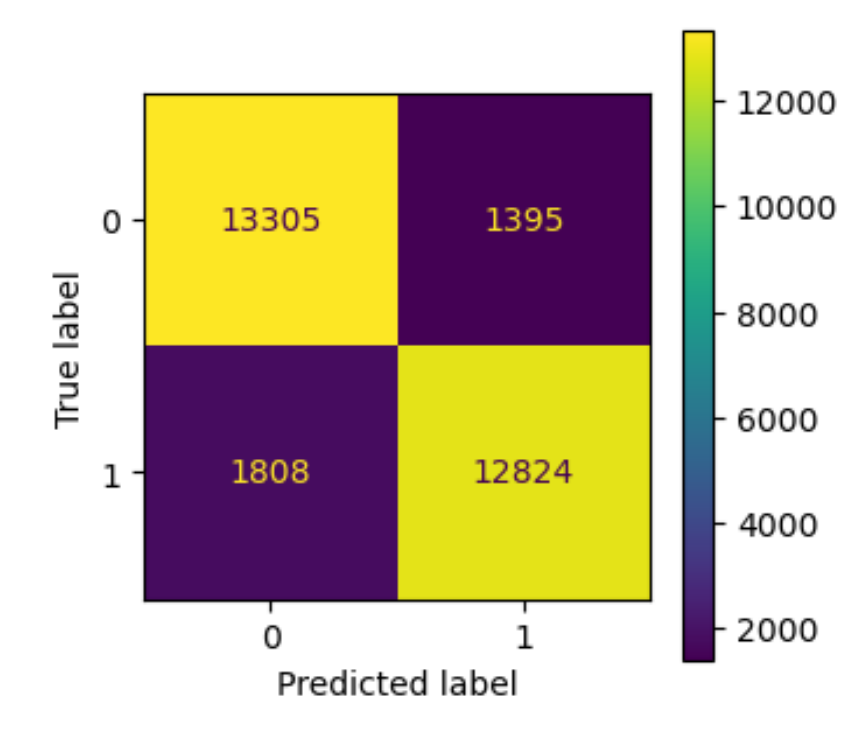
Confusion Matrix: Unsupervised

- NMF Performed the best

K-Means

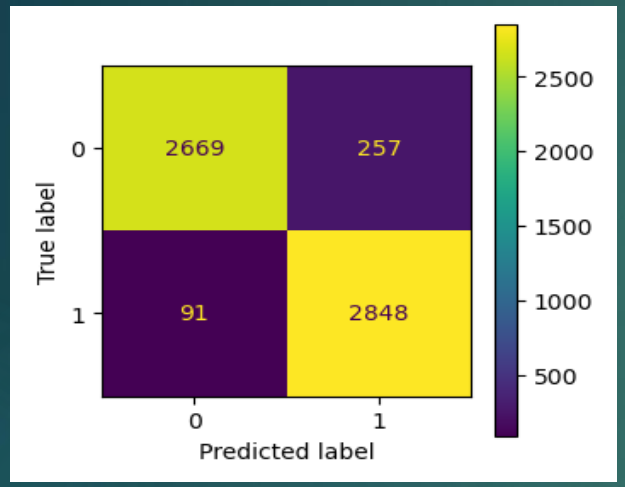


Non-Neg Matrix Factorization

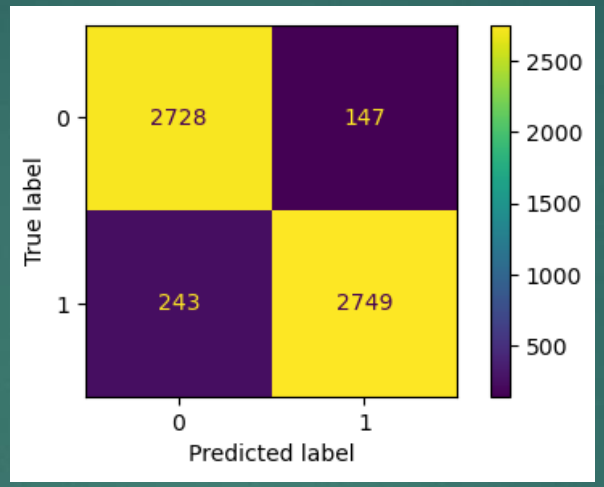


Confusion Matrix: Supervised

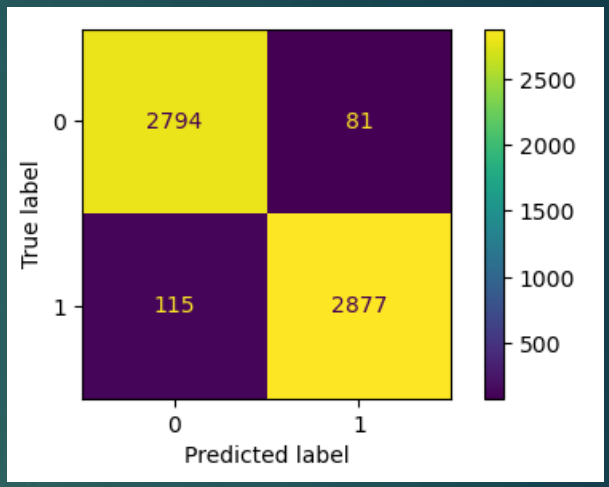
SVD



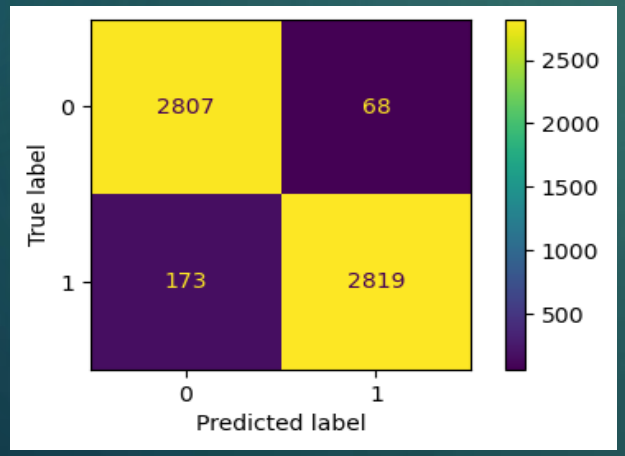
Random Forest



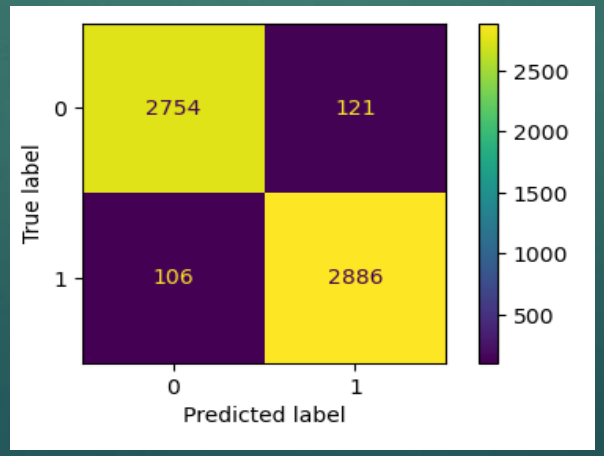
Support Vector Machines



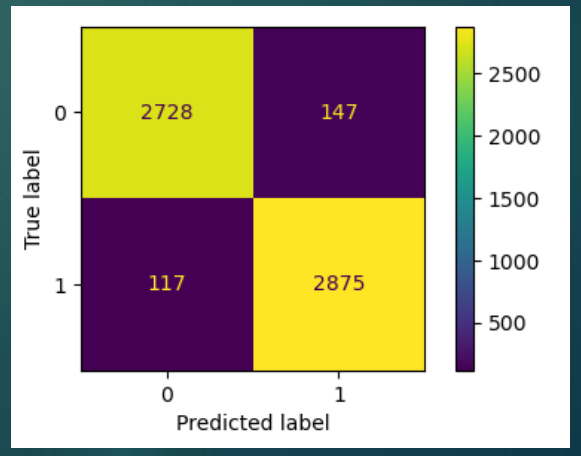
K-Nearest Neighbors



Gradient Boost



Logistic Regression



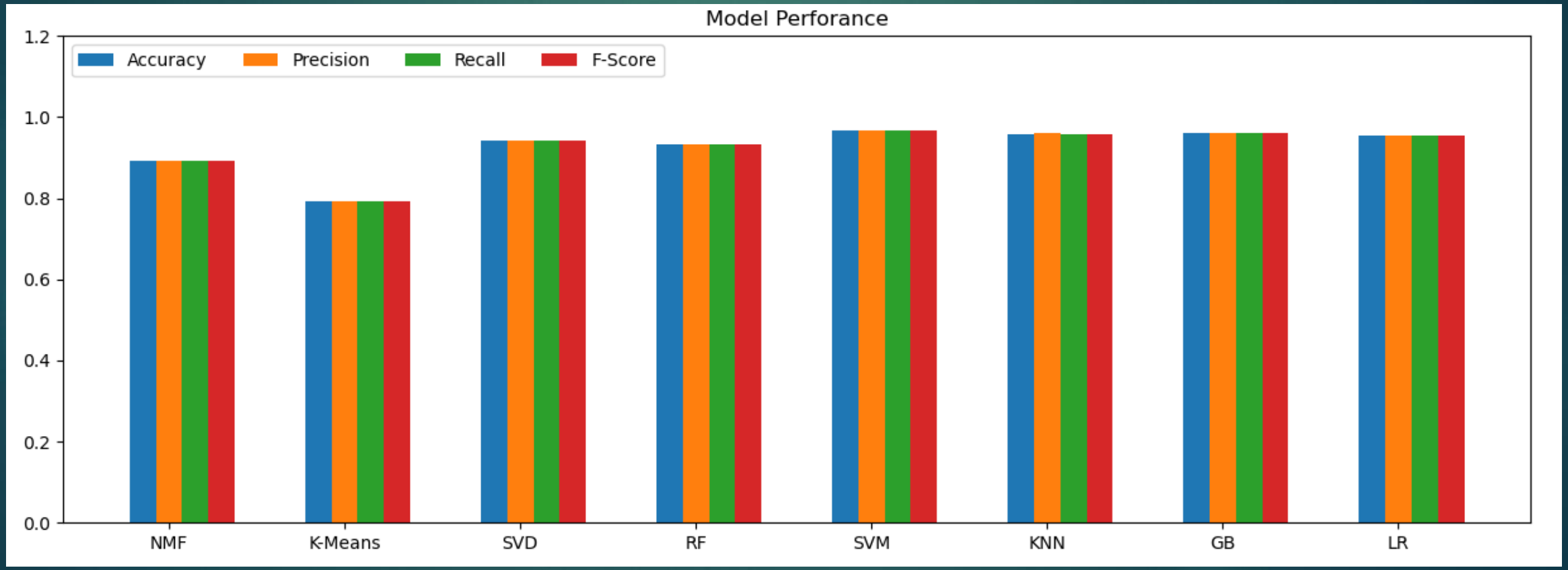
Accuracy and Execution Time

- While the Non-Negative Matrix factorization did perform well as an unsupervised method, with an accuracy of .891, this was 7.6% worse than the best Supervised learning method (SVM).
- However, the execution time of SVM was 34 times longer. Therefore, there is clearly a tradeoff between accuracy and speed.
- A "best of both" solution could be Logistic Regression. It performed quickly, and only suffers a 1% drop in performance from the best model.
- Unfortunately, labeled data would be necessary to train the model. Without labeled data, accuracy will take a hit based on this evaluation.

Method	Accuracy	Execution Time
NMF	0.891	2.07
K-Means	0.766	0.53
Singular Value Decomposition	0.941	2.56
Random Forest	0.934	1.77
Support Vector Machines	0.967	84.11
K-Nearest Neighbors	0.959	9.95
Gradient Boosting	0.961	71.72
Logistic Regression	0.955	2.44

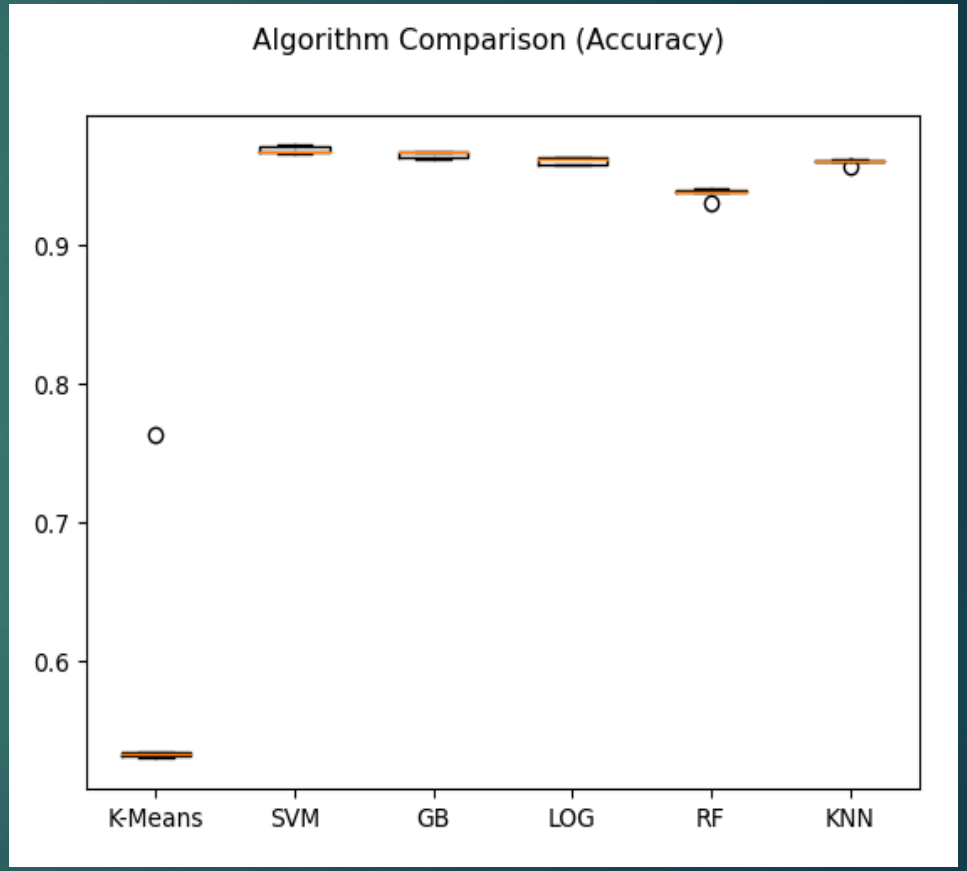
Accuracy, Precision, Recall, F-Score

- Accuracy, Precision, Recall, and F-Score were all nearly identical for this project. As the target data is balanced, this is to be expected. Unbalanced data could skew performance, and that would be recognized in metric such as precision and recall.



Cross Validation

- Interestingly, K-Means performed much worse when cross validation was used as the metric.
- As you can see, there is one iteration where the results are in the mid .7s; however, the boxplot shows that other iterations are below .6. This was not apparent from the previous tests.
- It is clear that SVM performed the best over the majority of its iterations.



Discussion

- Room for improvement in this project would be to train and test on a more realistic selection of data. This dataset was nearly 50/50 in malware and non-malware. However, it is unlikely that that is the true incidence of malware among the apps that are submitted to the app store.
- A point of learning for me during this project was the realization of the importance of cross validation. I have become used to splitting data 80/20 for training and testing. However, the cross validation showed that this can actually miss important results, based on the composition of that 80/20 split. Cross validation works across x iterations and averages the results. Based on those interations it is clear that my initial results for K-Means are actually skewed.

Conclusion

- This project strove to apply both unsupervised and supervised machine learning methods to the problem of classifying malware on the Android app store. Since labeling data can be laborious and time consuming, it was hope that an unsupervised method would perform as well as the supervised methods.
- While each of the 8 algorithms performed well for this task, the best accuracy results were achieved by Support Vector Machines and Gradient Boosting. However, both of these approaches needed far more resources to be trained. I believe that the best compromise model is *Logistic Regression*, as it trained and predicted quickly, and its accuracy was only 1% worse than the best performing model.