Comparing Classifiers: A Look at Machine-Learning and the Detection of Mobile Malware in COVID-19 Android Mobile Applications

Seth Johnson
sjohnson154@unomaha.edu
University of Nebraska at Omaha
Omaha, Nebraska, USA

Ray Donner
rdonner@unomaha.edu
University of Nebraska at Omaha
Omaha, Nebraska, USA

Alfredo J. Perez
alfredoperez@unomaha.edu
University of Nebraska at Omaha
Omaha, Nebraska, USA

ABSTRACT

The COVID-19 pandemic was a catalyst for many different trends in our daily life worldwide. While there has been an overall rise in cybercrime during this time, there has been relatively little research done about malicious COVID-19 themed AndroidOS applications. With the rise in reports of users falling victim to malicious COVID-19 themed AndroidOS applications, there is a need to learn about the detection of malware for pandemics-themed mobile apps.. In this project, we extracted the permissions requests from 1959 APK files from a dataset containing benign and malware COVID-19 themed apps. We then created and compared eight unique models of four varying classifiers to determine their ability to identify potentially malicious APK files based on the permissions the APK file requests: support vector machine, neural network, decision trees, and categorical naive bayes. These classifiers were then trained using Synthetic Minority Oversampling Technique (SMOTE) to balance the dataset due to the lack of samples of malware compared to non-malware APKs. Finally, we evaluated the models using K-Fold Cross-Validation and found the decision tree classifier to be the best performing classifier.

CCS CONCEPTS

• **Security and privacy** → *Malware and its mitigation*; • **Computing methodologies** → *Machine learning algorithms*; *Classification and regression trees*; *Neural networks.*

KEYWORDS

datasets, machine learning, neural networks, COVID-19, Android applications, Android, Malware

ACM Reference Format:
Seth Johnson, Ray Donner, and Alfredo J. Perez. 2023. Comparing Classifiers: A Look at Machine-Learning and the Detection of Mobile Malware in COVID-19 Android Mobile Applications. In *The Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '23), October 23–26, 2023, Washington, DC, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3565287.3617629

MobiHoc '23, October 23–26, 2023, Washington, DC, USA

ACM ISBN 978-1-4503-9926-5/23/10…$15.00
https://doi.org/10.1145/3565287.3617629

1 INTRODUCTION

When COVID-19 started impacting nations worldwide, it appeared to be an opportunity for developers who were capable and willing to create malware that was disguised as benign apps to take advantage of the panic that pandemic had created [6] due to the large increase in individuals working from home on personal machines. An increase in cybercrimes and malware distributed via phishing emails and other means (e.g., Short Messaging Systems (SMS), social networks) contributed to new attack vectors [6]. Trend Micro reported that "8,840,336 threats related to the COVID-19 were detected in the first half of 2020 with most of the discoveries occurring in April, coinciding with the pandemic's peak . . . these threats consist of emails threats, URLs threats, and Malware threats, which refer directly to the epidemic or indirect"[4]. According to Wang, nearly 75% of malicious app developers during the pandemic first utilized their certificates in January of 2020[5].

In this work, we study the relation between permission requests by an APK files in COVID-19-themed Android applications. By using the AV Rank, permission spread, and Total Permission Requests (TPR), we trained and tested four different classification algorithms to compare their ability to identify malicious APK files. The four models include Categorical Naive-Bayes (CNB), Support Vector Machine (SVM), Decision Tree, and Fully Connected Neural Network (FCNN) models. Additionally, we evaluated the use Synthetic Minority Oversampling Technique (SMOTE) to balance the training dataset because of the lack of samples of malware compared to non-malware. We plan to answer the following questions:

- **RQ1**: Which model do we believe is best for classifying malicious APK files?
- **RQ2**: How does the use of SMOTE affect the accuracy, precision, and recall of the models evaluated using K-fold cross-validation (CV) ?
- **RQ3**: Does considering the total amount of permissions as a feature APK impact the models' behavior?

2 RELATED WORK

In addition to compiling the dataset we utilize for our research, the Wang et al. team collected other information and characteristics about COVID-19 themed applications. such as origin and target region, duration of developer activity, the type of malware, frequency of obfuscation techniques, and frequency of APK repackaging/ app

replication[5]. Additional works should also be noted here in this field.

Notable studies in classifying and detecting malware in Android applications using Machine learning include the 2015 study by Almin and Chatterjee, who propose the Android Application Analyzer (AAA) [1]. Their research proposed a method in which they develop a user-interactive system that collects permission requests during install time to train k-means clustering models for malware classification, noting the flaws of signature-based techniques of pre-existing consumer Antivirus platforms. When compared to these platforms, the researchers find their approach is more efficient. However, they recognize that unknown malware types can break their cluster-based model for malware types with respect to permission requests, which can be a time-consuming process to rebuild said clustering [1].

Similarly, the DroidMat system [8] uses machine learning. DroidMat extracts permissions and app *"intents"* to infer various API calls, which are then passed to a k-Nearest Neighbors(kNN) classifier to identify an APK file as benign or malicious. In this research, DroidMat was compared with Androguard, and they found that DrodMat is more efficient than Andorguard. It notes two specific Android malware families that DroidMat has trouble identifying due to the use of installation of the malware payload from an external source, which circumvents classification.

Other works, in addition of using ML models to detect malware use permission requests. For example, Droid-Sec uses permission requests and API calls (static analysis) and runtime behavior collected via DroidBox (dynamic analysis), and behavior to compare a Deep learning network to Naive-Bayes (NB), Support Vector Machines (SVM), Logistic Regression (LR), and Multi-layer Perception (MLP), all optimized using a grid-search technique. They found that the deep neural network (200 features, 3 hidden layers, 150 neurons per layer) had the highest accuracy on a training set of 300 apps, and teste on 200 apps [9].

The 2013 study "PUMA: Permission Usage to Detect Malware in Android", by Borja Sanz also employs permission requests to compare the performance of Naive-Bayes, Random Decision Tree, and Random Forest (RF), which are all supported with a KFold CV with a 90/10 split. This study collects benign android apps off of Android marketplaces, and the malicious apps from VirusTotal's catalog. This study finds that the RF classifier is best by way of the Area Under Curve metric[7].

The 2013 study by Amos et al.[2] uses a database of 1738 unique apps to analyze six classifiers namely, RF, Naive Bayes, Multi Layer Perceptron (MLP), Bayes Net, Logistic Regression (LR), and J48, all evaluated with cross-validation. They found that, while many of these models are commonly applied to different techniques, they have relatively high False positive rates, even from their highest performing model, which was a Bayes Net. They concluded that feature vectors can vary drastically, and while they use a much larger dataset than those previous studies, they wanted to target applications from Google Play [2].

There are several aspects about the previously mentioned research. While more recent work deploys machine learning in various frameworks, many of these studies comparing classifiers were done before COVID-19, when there was no knowledge about how both benign and malign pandemics-related apps would have been implemented [5]. We intend to provide insight into this area, comparing classification models to statically detect COVID-19-related Android applications and further contribute detection frameworks for global emergencies-related apps.

3 METHODOLOGY

3.1 The Dataset

We used a dataset of 1971 APK files collected by Wang et al 2021 to train and test our classification models [5]. Their study correlated each APK file with a non-negative integer known as an "AV Rank" calculated by the online tool, VirusTotal. If it is greater than 0, at least one Antivirus software flagged the file as malicious [5]. Because of this nature, we use this as the labels set to train and validate our models. The features we considered were binary arrays representing the permissions requests extracted from an APK file, referred to as the "permission spread", and the sum of that binary array, or the "Total permissions requested", or TPR. Of that, 270 APK files were labeled malicious, or about 13.78±0.7% of the 1959 APKs that were successfully analyzed. If we consider Wang et al's dataset size of 4,322 total APK files with 611 malicious APK files out of that, they come up with a rate of about 14±0.5%. Thus, we confidently believe the dataset used throughout this study is a proper representation of the original one used by Wang et al. in their analysis.

During the pre-processing phase of our research, we noticed some characteristics worth pointing out. Referring to Figure 1, the frequency of malicious APK files, (represented by the 1 class in orange in the figure), have more spread out TPR values compared to the benign class (the blue, 0 class in the figure), which steadily drops off when the TPR > 10.

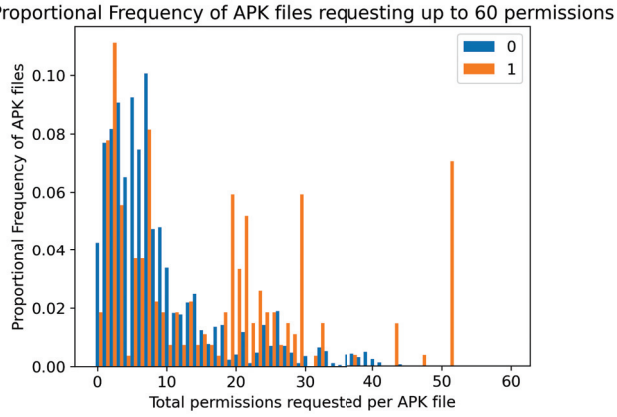

Figure 1: Proportional Frequency of TPR for each APK file.

The permissions requested by our APK files follow a similar to what the Wang et al 2021 study found, except our most frequently requested permission was android.permissions.INTERNET, as opposed to `android.permissions.WRITE_EXTERNAL_STORAGE`[5]. This can be observed in Figure 2. Again, APK files with an AV Rank > 0 are classified as 1/malicious.

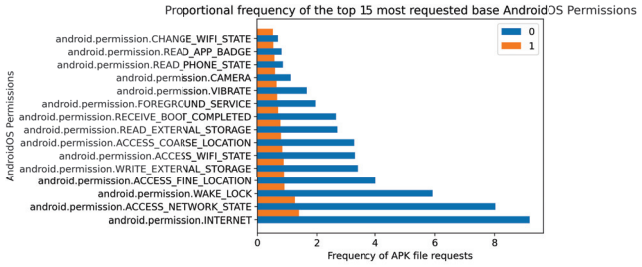

Figure 2: The top 15 most requested AndrodiOS permissions.

Because of the small percentage of malicious APK files within our dataset, we used the Synthetic Minority Oversampling Technique (SMOTE). SMOTE is an augmentative method that is used to address class imbalance issues, which is designed to alleviate class imbalance by creating synthetic samples for the underrepresented, or, minority class. It creates instances that are similar to already existing minority class samples effectively increasing the representation of the minority class in a dataset[3]. For our models, we used SMOTE to artificially increase the ratio of malicious APK files compared to the non-malicious files. This, in theory, should increase the model's sensitivity to the minority class which should help make more accurate predictions.

We utilize a Python script to compile our dataset `APKScanner.py`, which then is used to train and test our four classifiers with a Jupyter Notebook `Modelsv2.ipynb`. Figure 3 depicts the flow of execution for each of these files.

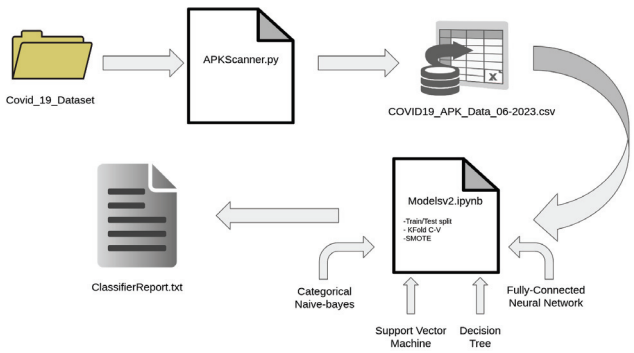

Figure 3: Process of code execution for this project.

`APKScanner.py` calls the Androguard Python module to unpack and extract the permissions from the APK files within the dataset. During this process, it stores the failed APK file hash and runtime error thrown in a text file available for further analysis. `APKScanner.py` then compiles a CSV file that consists of the APK file's 256 Hash name, the Application name, the package name, its AV Rank, whether or not another application uses the file, its TPR, and its permission spread. The data is then processed by `Modelsv2.ipynb`, and is executed twice. In each of the two runs, it calls each of the four models in four different groups that represent the four different data augmentation scenarios. Each group uses the same parameters for data preparation: `random_state=0` to fully randomize the order of our dataset and a `test_size=0.1` to maintain a 90/10 train/test split.

The first group executes a generic train/test split with an unmodified dataset, which serves as our control. The second group calls these classifiers again with a train/test split on a pre-processed SMOTE dataset. The third group executes KFold CV for each model, 20 folds each, on an unmodified dataset. For the fourth group, KFold CV is then executed again on the re-processed SMOTE dataset. Each classifier is called with the same parameters per training condition, and they all execute a binary classification: 0 if the APK file is benign, or 1 if it is malicious. Results are then recorded in a text file that records the average accuracy, F1-score, precision, and recall values for each model under each training condition. The results focus on the metrics for the malicious APK file classification. This script is then called a second time to train models off of a features dataset that does not consider TPR, but only the permission spread of each APK file. The performance is recorded in an additional text file.

3.2 Model Configurations

All models were executed with as close to default configurations as possible. We chose not to optimize each classifiers' parameters for this study. For the CNB classifier, we included an extra parameter, `min_categories=len(perms[0])` to manage indexation errors that occurred from uneven train/test split sizes when performing K-Fold CV with re-sampled SMOTE datasets. We also found that our SVM classifier would overfit our data if the kernel was set to "linear". The decision trees classifier naturally overfit the data, but passing `splitter="random"` in the declaration of the classifier mitigated this issue. The FCNN was a simple 3 layer network, with an input layer of size equal to the length of our features set, a hidden layer with 1959 units, and 2-node output layer.

4 EVALUATION

In order to evaluate our models, we prioritized recall, as it quantifies the number of correct predictions while also taking into account false negatives. It is most important that our classifiers correctly identify malware. Because of the threat malware poses, it would be better for our classifiers to identify benign software as malicious, as opposed to the other way around, so we prefer to have more false positives than false negatives. In addition to our evaluation, we also consider the F1-score since it's relationship to both precision and recall. The performance of each classifier is compared with these metrics across all groups and both runs to answer our research questions Run 1 can be visualized by Figure 4, and run 2 can be visualized by Figure 5.

4.1 Findings

Using these results, we note the following behaviors.

Categorical Naive-Bayes. This classifier had the most unique behavior out of all other classifiers. In groups 1 and 3 for both runs, it does not have any precision, recall, or F1-score to report, but does in groups 2 and 4. While this complicates our understanding of the classifiers behavior across all 4 groups, it does demonstrate the significant influence SMOTE has over this dataset. In run 1, group 2, it even obtains a recall value of 1.0, despite having the lowest

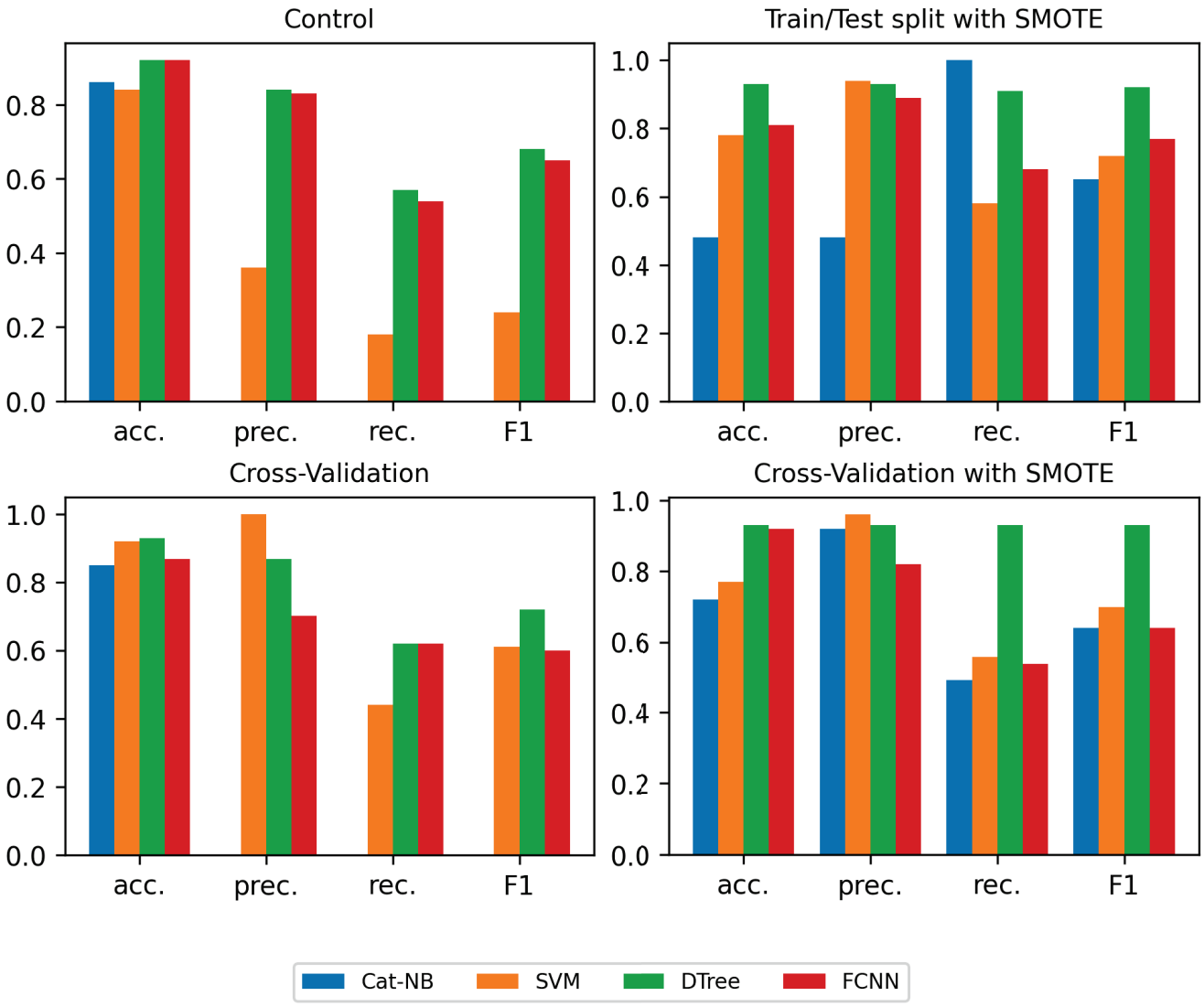

Figure 4: Classifier performance from run 1.

accuracy at 0.48. Run 2 saw all metrics slightly improve across all groups, but the behavior remained unchanged.

Support Vector Machine. While its recall value improved in groups 2 and 4 for both runs, the F1-score doesn't have the same significant improvement. This is likely due to its precision value. In run 1, the it held the highest precision value outside the control group, peaking at 1.0 in group 3. In run 2 it maintains the highest precision values in groups 1 and 3, even achieving 1.0 precision in the control group. However, precision drops in groups 2 and 4, being outperformed by the Decision tree.

Decision Tree. The decision tree consistently had highest recall and F1-scores in all groups across both runs, only having the recall value outperformed by CNB in run 1, group 2 . Being introduced to SMOTE improved the performance further in both runs, oscillating upwards between groups 1 and 2, and 3 and 4. The values stand out more in run 1 than run 2, but that's more likely attributed to the improvement of the other classifiers' performance when TPR was removed from the features set in run 2. That said, it does also benefit from the change to the features set

Neural Network. In run 1, the FCNN was the only classifier that didn't seem to have a consistent pattern to its behavior, giving us the impression that SMOTE didn't have a significant impact for this classifier. Run 2, however, shows a much more significant impact when SMOTE is applied. This gives us the impression that the FCNN is quite susceptible to changes in its features set. Regardless of that finding, it still swaps places with the SVM for having the 2nd highest values, even slightly dropping recall and F1-scores from group 3 to group 4 in the second run.

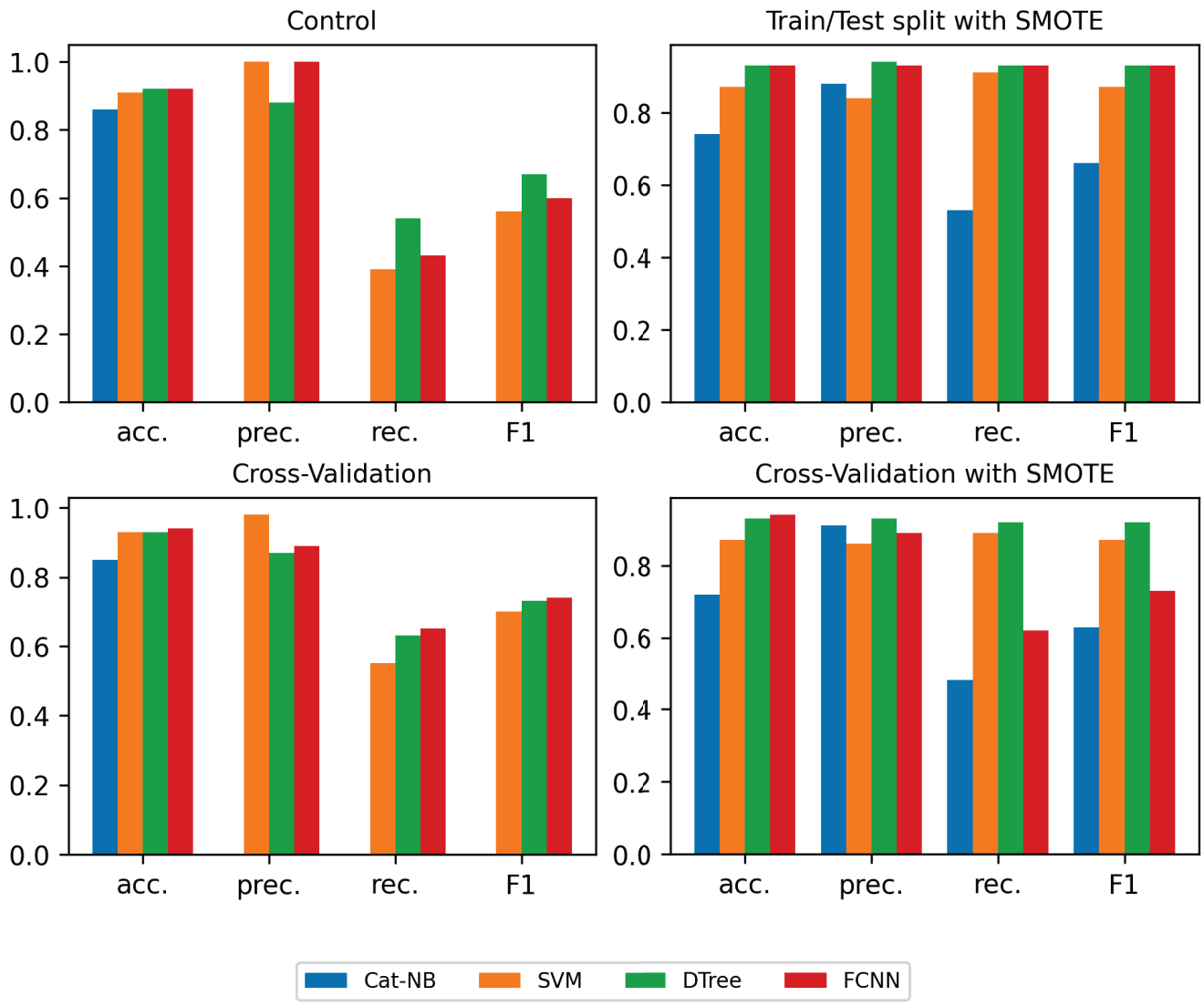

Figure 5: Classifier performance from run 2.

5 DISCUSSION AND FUTURE WORK

5.1 Research Questions

With our results, we can now address the questions motivating this research.

RQ1. The decision tree classifier was consistently the best performer. We also believe that the CNB is not a good classifier for our malware problem, given their overall performances across both runs. The high susceptibility to the balance of data-types in its dataset and the significant FCNN is also significantly impacted by the features due to this same fact but performs much more consistently regardless of the classification proportion. It comes close, but DT doesn't show the same susceptibility to varying feature sets and thus remains our preferred classifier.

RQ2. SMOTE seemed to have a significant impact on all classifiers, though the FCNN might be less likely to improve depending on its feature set. All classifiers saw noticeable boosts in performance in run 2, and all but the FCNN.

RQ3. Removing TPR from the feature set had a noticeable impact on the results of our models. We observe that performance metrics were comparatively higher on the second run with respect to the first. The FCNN classifier was most susceptible to this change, with it's changes in behaviors in groups 2 and 4 between the two runs.

5.2 Limitations

The most significant limitation was that when we executed the CNB models, the script did not output the precision, recall, or F1-scores for the malicious class in groups 1 and 3 for both runs. This had an

impact when we evaluated all of our classifiers to answer RQ1. Additionally, the FCNN threw warnings during its execution in group 4 for both runs, stating that the sampled class (1698 elements) was larger than the majority class (1605 elements). This behavior was not as detrimental, but could influence on the FCNN performance in that group. This, in addition to CNB's behavior in the groups where SMOTE was implemented, leads us to believe that the issue experienced with CNB's values to be a result of the proportion of our classes in the datasets that were used to train and test the classifier.

Additionally, our models were specifically trained our models for COVID-19-themed Android applications. As a result, our models could be not as effective when introduced to other types of applications, or deployed in different circumstances.

5.3 Future Work

We want our models to be useful to as many people as possible in the event another pandemic breaks out, so we hope to eventually apply our findings in the development of an Android application to assist in malware detection and alert an end-user on the risk of various apps and the permissions they request at install time. We recognize that in order to achieve this, we may need to broaden our model's capabilities in detecting malware beyond applications that are COVID-19 themed. We plan to implement an optimization of the parameters of the decision tree classifier to further improve its performance. Because we restricted the optimization parameters during our study, we feel that this step will further tune this classifier for a variety of more specific use cases.

Another way we would like to develop our research in the future is to modify our models to classify malware types. By observing the behaviors of the application, we could potentially determine the type of malware the application. This would enable our ability to detect and warn users of emerging threats more efficiently. Thus, by comparing the permissions to the malware type, we would then be able to train our models to identify types of malware more efficiently, and quite possibly receive better results when testing files for malware in general.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under grant award 2308741.

REFERENCES

[1] Shaikh Bushra Almin and Madhumita Chatterjee. 2015. A novel approach to detect android malware. *Procedia Computer Science* 45 (2015), 407–417. https://doi.org/doi.org/10.1016/j.procs.2015.03.170

[2] Brandon Amos, Hamilton Turner, and Jules White. 2013. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th international wireless communications and mobile computing conference (IWCMC) (IWCMC)*. IEEE, 1666–1641. https://doi.org/10.1109/iwcmc.2013.6583806

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357. https://doi.org/10.1613/jair.953

[4] Raghad Khweiled, Mahmoud Jazzar, and Derar Eleyan. 2021. Cybercrimes during COVID -19 Pandemic. *International Journal of Information Engineering Electronic Business* 13, 2 (April 2021), 1–10. https://doi.org/10.5815/ijieeb.2021.02.01

[5] Haoyu Wang Pengcheng Xia Yuanchun Li Lei Wu Yajin Zhou Xiapu Luo Yulei Sui Yao Guo Guoai Xu Liu Wang, Ren He. 2021. Beyond the virus: a first look at coronavirus-themed Android malware. *Empirical Software Engineering* 26, 4 (June 2021), 38. https://doi.org/10.1007/s10664-021-09974-4

[6] Bernardi Pranggono and Abdullahi Arabo. 2020. COVID-19 pandemic cybersecurity issues. *Internet Technology Letters* 4, 2 (October 2020), e247. https://doi.org/10.1002/itl2.247

[7] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. 2013. Puma: Permission usage to detect malware in android. In *International joint conference CISIS'12-ICEUTE 12-SOCO 12 special sessions (Advances in Intelligent Systems and Computing)*. Springer, 289–298. https://doi.org/10.1007/978-3-642-33018-6_30

[8] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia joint conference on information security*. IEEE, IEEE, 62–69. https://doi.org/10.1109/asiajcis.2012.18

[9] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2013. Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM (SIGCOMM)*. ACM, ACM, 371–372. https://doi.org/10.1145/2619239.2631434