MALICIOUS APP BEHAVIOR DETECTION

Eralda Caushaj, PhD, Assistant Professor | Varun Vikram, GRA, MSIT

College of Business + IT, MSIT Program

INTRODUCTION

The Android mobile operating system is phenomenally popular. As of December 2017, the total number of smartphone users globally is about 4.77 billion and Android commands 80.0 percent of the total market share. There are currently around 2 million apps available in Google Play, the official Android market. Our dataset includes 1,869 records in total and six features as 1.) Dangerous combo, 2.) Number of ads, threats, 3.) Perm vs average, 4.) Over privileged app permissions, 5.) In app purchase and 6.) Run at startup. The dataset has three classes: benign, safe and malicious, with maximum number of instances for benign class (78.49% of the whole dataset). Through our research, we are interested in finding an accurate classification for the minority classes (safe, malicious). To overcome this imbalance dataset problem, we have implemented SMOTE (Synthetic Minority Over-sampling Technique), an over-sampling approach in which the safe and malicious classes are over-sampled by creating "synthetic" records. We use three supervised classifiers such as Naïve Bayes, SVM and Classification Tree. All classifiers were tested using WEKA , a data mining software.

Feature Selection

The following is the list of features that we use as input in our machine learning model to classify apps as malicious, benign, or safe. The features were extracted from analysis of our large data collection.

1. App permissions that each app requires. On our dataset we have identified 118 unique app permissions and each one of them is used as feature in the dataset. Some examples of app permissions include "View Network Information", "Location", "Storage", etc.

2. Dangerous combinations of permissions: Which combinations of permissions are dangerous was determined by examining the permissions of 20 high-risk apps with many permissions. The combinations of permissions that were causing security risks in those apps were used to create a list of combinations our model considers dangerous. The dangerous combinations are calculated based on the list of app permissions an app request and the potential risks each app permission can expose the user. A score is used to calculate it.

3. Number of ad networks associated with the app: We identified 79 different ad networks, and found that the maximum number of ad networks associated with a single app in our database was 13. We found that apps that contain many ad networks can pose more threats to a user's security, because their information can be leaked to many different advertisers.

4. Total number of privacy threats posed by the app: We also take into account the total number of threats posed by the app. By mapping a list of the top Android adware and malware from each quarter in 2014-2015 to the permissions they require to operate, we were able to determine how many threats each permission exposes a user to.

5. How many permissions the app has compared to the average of other apps in its category. Granting a large number of permissions to a single app can be dangerous to a user's security.

6. Requests for unnecessary permissions: we considered whether the app contains any permissions we know to be unnecessary to the app's function. We consider In-App Purchases and Run at Startup.

Machine Learning Models

We used three supervised classifiers such as Naïve Bayes, SVM and Classification Tree. We also used 10 fold cross-validation for our model in all the three classifiers. Cross-validation, is a model validation technique, mainly used in prediction, and estimation of how accurately a predictive model will perform in practice. In this validation technique, the dataset is split randomly into 10 partitions. We then fit our model to a dataset consisting of 9 of the original 10 parts which is also known as training data, and use the remaining portion for testing. This process is repeated 10 times and average over the whole process is taken. To overcome the imbalance dataset problem, we can use several techniques on data, by changing the distribution of imbalanced datasets. We used Synthetic Minority Over-sampling Technique (SMOTE) an over-sampling approach in which the safe and malicious classes are over-sampled by creating synthetic records. Also, we used "Randomize" filter to randomly shuffle the order of instances passed through it. When the SMOTE technique is applied, the new synthetic instances are generated at the end of the original dataset.

Classification Tree

A classification tree classifies new data points by using their various features to move down through a hierarchal decision structure known as a tree. A tree is composed of a series of nodes. At each node, a feature is compared against a set threshold. A classification tree is built by systematically creating nodes that reduce entropy by the greatest amount. Entropy is a measure of how disordered a data set is. The ideal entropy of 0 would indicate that a set is perfectly separated into correct classes. Classification trees are very prone to overfitting to training data, and thus must be "pruned" to ensure generalizability. Pruning involves limiting the size of the tree in various ways.

Naïve Bayes

The Naive Bayes model calculates how probable it is that a new data point belongs to a given class by looking at each feature and determining the probability of the class given the value of the feature. The features are assumed to be independent of each other. These probability values are calculated for each feature then multiplied together. The product is multiplied by the overall probability of the class occurring divided by the overall probability of each feature taking the given values to get the overall probability of the point belonging to the class. We used "Discretize" option for Naïve Bayes. Discretization is typically used as a pre-processing step for machine learning algorithms that handle only discrete data. In addition, discretization also acts as a variable feature selection method that can significantly impact the performance of classification algorithms.

Support Vector Machine

A support vector machine (SVM) calculates an optimal separating hyperplane between two classes. It uses the training points that are closest to the gap between the two classes, called the support vectors, to find the hyperplane that separates the classes with the largest margin. This hyperplane is then used as the classification boundary. Every point on one side of the hyperplane is given label 1 and every point on the other is given label -1. The particular implementation of SVM we used is C-SVM. SVMs also usually require scaling to operate effectively. However, SVMs handle high-dimensional feature spaces and complex decision boundaries with ease.

Results

Results for all the three classifiers (Classification Tree, Naïve Bayes, Support Vector Machine) are shown for both, without using SMOTE technique and with using SMOTE. AUC, CA, Precision, Recall, and F-measure are taken on average for all the three classes, benign, safe and malicious.

Results of Machine Learning Model Testing without SMOTE

Method	AUC	CA	Precision	Recall	F-measure
Classification Tree	0.943	0.961	0.961	0.961	0.961
Naïve Bayes	0.982	0.92	0.942	0.92	0.927
SVM	0.924	0.960	0.960	0.960	0.959

Results of Machine Learning Model Testing with SMOTE

Method	AUC	CA	Precision	Recall	F-measure
Classification Tree	0.984	0.971	0.971	0.971	0.971
Naïve Bayes	0.994	0.951	0.954	0.952	0.952
SVM	0.977	0.976	0.976	0.976	0.976

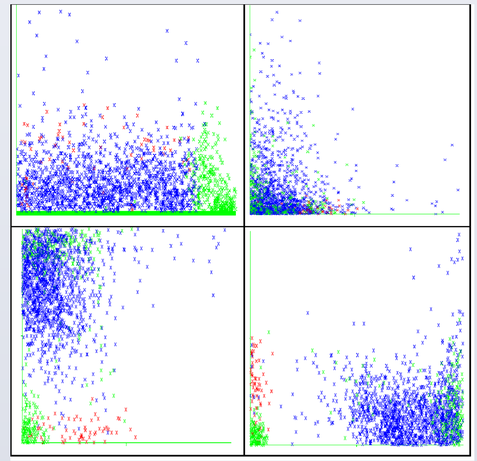
Based on the above results, libSVM on an average is performing better. Its setup allows it to take relationships between multiple features into account, and it constructs a nonlinear decision surface to ensure the best fit for our data. It is convenient to implement as well, as LIBSVM includes open-source implementation in a variety of languages including Java, the native programming language of Android apps.

Feature Contribution

This includes the features which are actually contributing more towards the final models. After comparing all the three models, below is the table showing features (all of them are present in 10 fold) contribution:

Attribute	Classifier Attribute Evaluation
Threats	1
In app purchase	2
Number of Ads	3
Run at Startup	4
Perm vs Avg.	5
Dangerous Combo	6

Feature Visualization



Conclusion

We used our data to identify 200 features which can be used to classify apps into safe, benign, and malicious categories. We have also demonstrated the results of classifying apps using these features. An SVM model is an effective way to categorize apps based on our features, managing to achieve 96% classification accuracy.

References

W. Enck, M. Ongtang and P. McDaniel, "On Lightweight Mobile Phone Application Certification", Proceedings of the 16th ACM Conference on Computer and Communications Security, 2009

Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. NDSS, 2012

B. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Android permissions: A perspective combining risks and benefits. In SACMAT '12: Proceedings of the seventeenth ACM symposium on Access control models and technologies. ACM, 2012

H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru and I. Molloy, "Using Probabilistic Generative Models for Ranking Risks of Android Apps", Proceedings of the 2012 ACM conference on Computer and communications security, pp. 241-252, 2012