June 30, 2023

Project Report

Introduction To Machine Learning

Binary Classification: Is a file malicious or not?

Adi Finkelstein 318863693

Raz Ronen 208279810

The Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

Exploratory Data Analysis

The analysis begins with a preliminary exploration of the dataset, examining the dimensions of the train dataset, which consists of 60,000 rows of file examples and 24 columns of features. These features include the target variable (label) and a unique identifier of the samples (sha256).

We identified duplications within the dataset and removed them to prevent bias. Additionally, we performed a feature analysis to gain a better understanding of the train dataset. This analysis aimed to uncover patterns and correlations among the features to enhance our insights:

Based on a **descriptive statistics analysis**, we discovered that the dataset comprises a combination of categorical and numerical features. While all the numerical ones are floats (except for one - size, which we'll convert to float later), “file_type_trid” exhibits 89 unique values, indicating a diverse range of file types within the dataset. Another categorical feature, "C," demonstrates 7 unique values, implying distinct categories or classes associated with this feature.

We can also identify few binary columns (the columns have 3 unique values in total since 1 is for the NULL values): “has_debug”, “has_relocation”, “has_resoucres”, “has_signature”, “has_tls”,” labels”.

At this point our dataset is balanced and we got 50% of malicious and 50% of benevolent files. We want to analyze each feature distribution, so we did some visualizations:

According to the **binary features plots** we conclude that files without debug (“has_debug”=0) are more likely to be malicious, while files with debug (“has_debug”=1) are more likely to be benevolent.

From plotting **15 most common** file types (categorial feature) by label we can tell that if a file type is “DOS Executable Generic” it most likely to be malicious but if it’ s “Generic .NET DLL/Assembly” or “WinRAR Self Extracting archive (4.x-5.x) the file is probably benevolent. But that’s just 3 file types out of 15 most common that we presented so it’s hard to determine (yet) if this feature is a good indicator to the label.

From plotting the **numerical features**, we can see that in most columns the peak forms at 0 and the graph is skewed towards the right side. It means that we have outliers on the right side of the plot in these columns, we will handle it in the next sections.

On the other hand, we can see that the “**A”** feature distribution looks like a gaussian, very similar to normal distribution (with slightly shifted towards the left) so **normally distribution will be assumed**.

While trying to see the “Size” and “Vsize” column's distributions, we found out it's hard to understand the distribution since the columns are not scaled but after applying the mathematical manipulation **Log**, we got clearer picture of the distribution.

We noticed that the mean value of benevolent files at some features (“exports”, “symbols”, “paths”, “urls” and “registry”) is higher than the malicious files, and the confidence interval for the mean is also higher for the malicious files. It means that these features might be a good indicator for the label.

We assume correlation between some features after creating **correlation matrix**: “size**”** is correlated with “numstrings”, “MZ” and “printables” while “avlength**”** is correlated with “printables”.

After analyzing feature correlation to each other we want to examine **feature correlation to the label**. To do that we used another graph that might be helpful for feature selection later. The graph indicates a positive correlation between the “B” feature to the label and a negative correlation between the “has_debug” & “has_signature” features to the label.

Pre-Processing

After splitting the train dataset into a **train** set and a **validation** set, we proceeded to explore the distribution of **null values**. Initially, we considered removing examples with missing values to prevent bias in our model. However, this approach resulted in the exclusion of over 50% of the dataset. As a solution, we decided to fill in the missing values using other features that exhibited a strong correlation to the value that missing.

Next, we identified the files that still had missing values in features that displayed a good correlation with the label. These files were then removed from further analysis. For the remaining features that were less correlated with the label, we considered three different methods to replace the missing values: median, frequency and KNN imputation.

We used visualizations to choose the most suitable method for our train data distributions. For example, when examining the "A" feature, we observed that using the most frequent value method disrupted the distribution and made it less normal. On the other hand, employing the median method adds a peak to the distribution. Ultimately, after applying the KNN method, we found that the distribution it seems like the distribution hasn't changed much.

That's way we chose to use KNN for **filling NULL** values of the not correlated numerical features. We noticed that Boolean features acts weird using the KNN method, so we will use Frequent value for those features.

Since ML models does not work with non-numeric values, we will **encode all the categorical features**. "C" has 7 kinds of data that has no relation or order, so we will use One Hot Encoder. On the other hand, "file_type_trid" has 89 different values and most of the values are using only 15 types. This is why we will group the bottom 1% as "Other" and use frequency encoder + one hot encoder.

We chose to use "BOXPLOT" (IQR) for analyzing the **outliers** in the numeric features in the dataset. We chose to use that method despite "A" us the only feature that normally distributed because we want to get some more information (like the mean) about the numerical features.

According to the boxplot graph we assume that Both "A" & "B" columns are close to normal distributed (since the median is in the middle of the box), but they both have a lot of outliers.

To handle the outliers, we will try to use the definition of an extreme outlier to remove these values:

The IQR method involves calculating the IQR, which is the range between Q1 and Q3. Any data point that falls below Q1 minus 1.5 times the IQR or above Q3 plus 1.5 times the IQR is considered an extreme outlier. We will implement a function that calculates how many outliers are there in each column. Then, we will remove only rows that contains more than a threshold of outliers.

We used winsorize where there are more than 5% outliers that was detected by IQR (because we do not want to remove significant amounts of our data).

The data is not normalized, as the ranges of all the numerical columns are different. However, **normalization** is going to prevent bias, enhance model performance and enable us to explain the model results according to the classification domain. We will try 2 methods to normalize the data set: minmax scaler and standard scaler. We chose to use the standard scaler since we want to apply models that assume the features are normally distributed.

After performing the pre-processing steps, we observed a change in the **balance** of our dataset, which was initially balanced. This alteration has raised concerns about the potential impact on the performance of our classifier.

We used the Synthetic Minority Oversampling Technique (**SMOTE**) to deal with that. This technique generates synthetic data for the minority class. SMOTE works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors. The disadvantage of this technique is that it increases the likelihood of overfitting since it replicates the minority class samples.

To deal with the **dimensionality** problem (that can imply overfitting, high costs and decreased model transparency) we first remove highly co-linear features to reduce model complexity by decreasing the number of features to increase model generalization. We remove features that have a correlation coefficient above a certain threshold with each other (not with the label because we want variables that are highly correlated with it). Then we are Implementing PCA for reducing the complexity of the dataset.

We checked the curse of dimensionality by comparing the number of features before and after applying Dimensionality Reduction Technique.

Classification Model Implementation

For **KNN** we used grid search to define the **number of neighbors** to consider when making predictions (small value of *k leads to overfitting* as it considers only few points. Also, it can be easily influenced by outliers. Even very large value of k leads to high error as it starts to take more global structure and running out of its boundary) generally, larger values of *k* reduce effect of the noise on the classification but make boundaries between classes less distinct. Furthermore, **weights** = 'distance' assigns weights proportional to the inverse of the distance from the query point. The 'manhattan' **metric** calculates the distance as the sum of absolute differences between coordinates, which can be more robust to outliers or irrelevant features. This can help reduce variance but may introduce some bias by assuming that all features contribute equally to the distance calculation. The grid search chose brute algorithm that can add bias to the model.

For **MLP** we used grid search to choose different configurations where the MLP model has a single hidden layer with 10, 15, or 20 neurons (too many neurons in the hidden layers may result in overfitting). We also got that logistic activation function is the most suitable for our needs. Furthermore, Increasing the alpha value can help prevent overfitting by penalizing large weights.

For **decision tree** we use the grid search to choose max depth, while a higher value allows the tree to make more complex splits, potentially capturing more relationships in the data. However, increasing max_depth can also lead to overfitting if the tree becomes too deep and captures noise or irrelevant patterns in the data).

For **random forest** we use the grid search to choose the number of trees hyper parameter (**n_estimators**): enlarging the number of trees in the forest tends to decrease the bias and increase the variance. Max depth: Increasing the maximum depth can lead to lower bias as the trees can capture more complex patterns in the data.

Evaluation

We choose to mostly evaluate our models using accuracy score that we want to maximize (TP+TN/TP+FP+FN+TN), precision (we also want to maximize) AUC and differences between the validation and training sets to check over fitting.

We also made a confusion matrix for each model: confusion matrix includes 4 sections. In every confusion matrix we want to get big count of True positives and True negatives and small number False positive and False negatives

In the random forest model, we got True positives 3957, True negatives 3997, False positive 475 and False negative 578 with 0.88303 accuracy and 0.88 precision.

The train AUC is 0.9731641973668121 while The test AUC is 0.957113764885035, which is good since it's pretty close but not the same (as expected).

In our specific confusion matrix, we observe a significant reduction in false positives and false negatives, along with a higher number of true negatives and true positives. Its indicates that our model can effectively distinguish between different labels, although it may not be perfect. Overall, the model performs well in most cases.

We compare between the different models using the MAE metric.

The MAE is the average absolute difference between the predicted and actual values. It gives us an idea of how wrong the predictions were.