Project 4:

CryptoGuard: Unveiling Ransomware Transactions

Jakob Long, Ethan Colley

COE379L

Introduction and Problem Statement:

For our final project in COE 379L, we have elected to evaluate advanced classical algorithms for tabular data. More specifically, we would like to investigate the performance of these advanced algorithms to the performance of the algorithms performed in project/unit 2. We want to see if the more advanced methods perform better overall or if a user can just stick to the more simple classification methods, additionally the testing of AutoGluon's tabularPredictor package is to be looked at as well.

We will do this by using a dataset on Bitcoin ransomware, and using each method to predict if a transaction is actually ransomware. In addition to comparing advanced methods to simple methods, we will also comment on which method works best overall and which performs the worst.

Data Sources and Technology Used:

As stated briefly above, for this project we will be using a Bitcoin heist ransomware dataset from the UC Irvine Machine Learning Repository. The dataset has 10 categories, all numerical except the Bitcoin ID and the ransomware label. The other columns include year, day, length, weight, count, looped, neighbors, and income. This dataset acts as a good dataset for classification and it actually has over 2.9 million instances. For the purpose of the project, we will need to drop the address column as it does not matter, and convert all ransomware instances into a single classification called ransomware. We will also need to convert some numerical

variables like day (it is out of 365, converting these 365 days into larger categories like months would be more beneficial) into larger categories for easier implementation and understanding of our results. We have also decided to even out the dataset, as there are substantially more ransomware cases than non ransomware. To remedy this, we dropped a large amount of the ransomware data and made it to where the number of ransomware and non ransomware cases were equal. This still left us with about 76 thousand cases which is more than enough.

The technology we will be using during this project will primarily be python(.ipynb) in Jupyter notebook. Within this script we will be using multiple libraries such as pandas, numpy, scikit-learn, etc… All of which can be seen within the Jupyter notebook for further details regarding the specific libraries that were used.

Data Pre-Processing:

In order to properly test the models we're investigating and their classification capabilities, we needed to properly pre-process the data. To do so, the data was manipulated in the following ways.

Firstly, the data types of each variable that were present within our original dataset. Of which we found three object types, and six numeric types. From this we were able to determine that the address variable wasn't necessary for our model, as it's simply the name of the BitCoin transaction. In addition to this the day variable was converted to a month variable as a way to reduce the one-hot encoding that would occur later. Following this, the income variable was changed from Satoshi to BitCoin as a way to properly represent our end goal. Lastly, we changed all the different types of ransomware to fall under a singular category of ransom, with non-ransomware transactions having the label of white. Visualizations of the data prior to one-hot-encoding are available in the Jupyter notebook, and are not included here for sake of duplicity.

With the dataset now properly set, it was found that there was a large class imbalance between the 'ransom' and 'white' labels in our data. This poses an issue as the label variable is our target variable. As a way to correct this, we equalized the two sets to have a better balance between the two variables. This left us with only 81,000 values from a staggering 2.9 million

original observations. However, this was necessary to not overfit our models with data that wasn't affected by a ransom.

Following this, the data was then visualized. Of which we were investigating the distribution of each variable within the dataset. Where we found that the categorical variables are normally distributed, as shown in images 1 and 2.

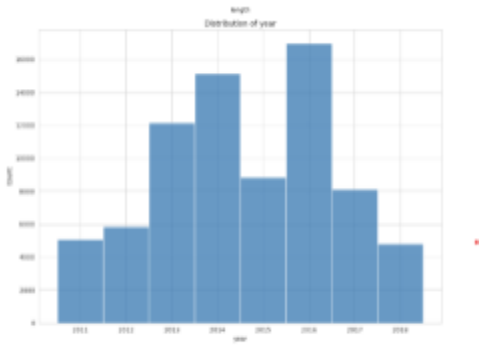

Image 1 - Distribution of Year

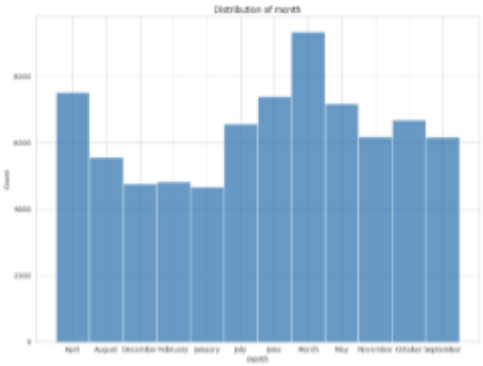

Image 2 - Distribution of Month

After visualization, one-hot-encoding was performed on all the categorical variables. This was to ensure that the data could be properly accepted & used by our methods that were being tested. This however, would not work for the AutoGluon package for tabular machine learning models. We performed all of the same steps as mentioned prior, but left out one-hot-encoding. This is due to the package solver being able to accept data in that format & properly handle categorical data, provided they're given a label name of your target variable.

Methods Implemented:

For the purposes of this project, we have identified 7 possible methods that we would like to use in the implementation of our project. We have 5 methods that are considered our simpler methods that we learned in unit two, and those are the decision tree, random forrest, k nearest neighbors, Naive Bayes, and logistic regression. The more advanced methods we would like to test are the supportive vector machines (SVM) and TabTransformer.

For all of the normal classical models, the implementation of the methods was fairly straightforward. For every model except KNN, no additional work had to be done other than importing the models from scikit-learn. For the K nearest neighbors model, after importing the K nearest neighbors method from scikit-learn, we had to instantiate the model with its hyperparameter of 3 and then check the accuracy on both the training and test sets, which is exactly what we did for project two.

For the advanced classical methods, implementation of the supportive vector machines, both normal and linear, is just a matter of importing the methods from scikit-learn and implementing them as we did the lower level classical methods. Implementation of the tabular transformer requires a bit more work than just importing it from scikit-learn.

The implementation of the TabTransformer model was quite complex. As there was a robust ANN architecture that required the input data to be split, then concatenated after the categorical variables had been correctly processed. This architecture is shown in figure 1.

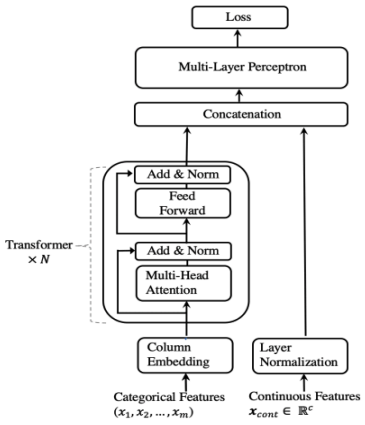

Figure 1: The architecture of TabTransformer.

From this figure we can see that the categorical features must be processed, firstly being turned into numerical representations so that the model can better understand the categorical variables. The Transformer architecture is then applied to categorical data that has been converted, where the MultiHeadAttention allows for the model to process the different categories all at once & weigh the importance of the different elements. Unfortunately, our model doesn't implement this due to issues arising with the model fitting and throwing an error that was too complex despite referring to documentation. Despite this, we still had self-attention layers that helped the model capture the true relationships. Following this, the numerical variables & now newly configured categorical data are concatenated, and pushed through a dense ANN layer, that is then finally trained under supervised learning.

A final note on our methods is that upon reading one may realize that we did not mention the random forrest method. After implementing and testing this method, the run time proved too large, likely due to the large number of datapoints that we are predicting. All other methods ran in just a few minutes at the maximum, while this method ran for nearly an hour with still no results, so we decided that the time trade off with the model was ultimately the equivalent of a bad result.

Results & Analysis:

Classical methods:

Decision Trees -

The results of the decision tree model show that the accuracy of this model on our dataset was 83% with a recall of 84% and a precision of 84% on the test set. It had 1978 false positives and 1940 false negatives. Compared to the size of our dataset these false positive and false negative numbers are considerably low.

Logistic Regression -

The logistic regression model performed a bit worse than that of the decision tree with an accuracy of 59%, a 60% recall, and a precision of 61%. There were also 4866 false positives and 4634 false negatives, so we can safely say that this will not be our best performing model.

KNN -

The accuracy of the KNN set is better than logistic regression but worse than the decision trees, with an accuracy of 73%, recall of 71.9%, and precision of 75.6%. This model had 3440 false positives and 2846 false negatives.

Random Forrest-

As stated above, this method took substantially longer than any other method and still did not finish. While this method may yield good results, they would ultimately be hurt by the over one hour run time while other results can be found within a matter of minutes.

Naive Bayes -

The Naive Bayes method had a 62% accuracy, a 36% recall, and an 84% precision. There were 7798 false positives and 862 false negatives in this method, an odd skew compared to the other methods.

Advanced Classical methods:

Support Vector Machine -

The first of our more advanced methods was the normal support vector machine, which yielded an accuracy of 56%, a recall of 88%, and a precision of 55%. This model also yielded 1473 false positives and 8746 false negatives, a bit of an opposite skew from Naive Bayes.

Linear Support Vector Machine -

The linear support vector machine had an accuracy of 69%, a precision of 58%, and a precision of 79%. This method had 5198 false positives and 1903 false negatives.

TabTransformer -

The TabTransformer model had an accuracy of 83%. This performance was only from 50 epochs, with no clear signs of overfitting, so it's likely the overall accuracy & performance of this model could be improved further by implementing more epochs for the model.

AutoGluon -

The AutoGluon solver tested an additional nine models not tested by us! This provides some insight into the performance of other advanced models such as XGBoost, without our own self implementation. Of which it was seen that all of these models performed quite well, with a high of 87% being the WeightedEnsemble54grt_L2 model and 76% being the NeuralNetFastAI model.

Conclusions:

In conclusion, we have taken an incredibly large Bitcoin ransomware data in order to see if our classical models can accurately predict the ransomware. Since our dataset had over 2.9 million instances, we condensed the dataset significantly and did type conversions on some columns to better suit the classical methods. Once the data was split, we tried several normal classical methods and several more advanced methods to determine which classical method had the best performance. Ultimately, a model from the AutoGluon machine learning tabular data package, WeightedEnsemble54grt_L2 method had an 87% accuracy and was our top model which is not unexpected since it is considered a more advanced method. In addition to being quite robust with more skilled engineers behind the implementation. More surprisingly, there were two models tied for second best performance, one of which is the advanced TabTransformer method, and the other being the non advanced decision trees method, both having an accuracy of 83%. This is interesting as the transformer model would likely outperform the simple Decision Tree method, if the number of Epochs were to be increased, or if the MultiHeadAttention layer were to be properly implemented.

References:

Support Vector Machines:

https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

https://scikit-learn.org/stable/modules/svm.html#

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

TabTransformer Information:

https://paperswithcode.com/method/tabtransformer

https://keras.io/examples/structured_data/tabtransformer/

https://www.kaggle.com/code/antonsruberts/tabtransformer-w-pre-training/notebook

Data Origin & Tidying:

https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

https://auto.gluon.ai/stable/tutorials/tabular/tabular-quick-start.html

AutoGluon:

https://auto.gluon.ai/stable/tutorials/tabular/tabular-quick-start.html