Project Report for ”Save the Attack”

K Yashovardhan Reddy
International Institute of Information Technology, Bangalore
IMT2019097
Email: yashovardhan.reddy@iiitb.ac.in

Kalahasti V V Deepesh
International Institute of Information Technology, Bangalore
IMT2019508
Email: deepesh.vv@iiitb.ac.in

I. Introduction

SecurityPlus.ai is an upcoming computer security software company. The existing products of the company provide total malware protection for systems but a new research team of the company is focusing on a different problem statement of predicting the possibility of a system getting infected by malware. You are a newly hired ML Engineer who has been assigned the task of working on this challenging big data problem. The dataset for the above has been collected by monitoring users' systems who have availed the services of SecurityPlus.ai. Each row in the data corresponds to one machine and includes the machine information along with other threat information which includes Engine Version, Processor, OS version, Firewall etc.

The goal of this project is to predict whether the given system will be infected by Malware based on certain information about the machine that is given to us. Each machine is uniquely indentified by a 'Machine identifier' and we need to predict the values in the column 'HasDetections'. This column can take either '0' or '1'. A value of '1' indicates that malware has been detected on the Machine and '0' indicates no malware.

We have to use the values in traindata.csv to train our ML model and predict the values in testdata.csv. The output will be scored based on ROC-AUC curve score.

II. Dataset

The dataset hosted here is a part of competition where we have to detect Malware on devices. The training data has 7137943 rows and 83 columns, out of which 29 are categorical and 53 numerical columns. The target column is named 'HasDetections' whose values are binary, and tell us whether the device is affected by the malware or not. The dataset obtained after preprocessing was almost balanced with similar frequencies for 'Has Detections' and 'Does not have detections' as shown in Figure:1. This is a binary classification problem as there are only two classes for the output variable. The link to the dataset is here.

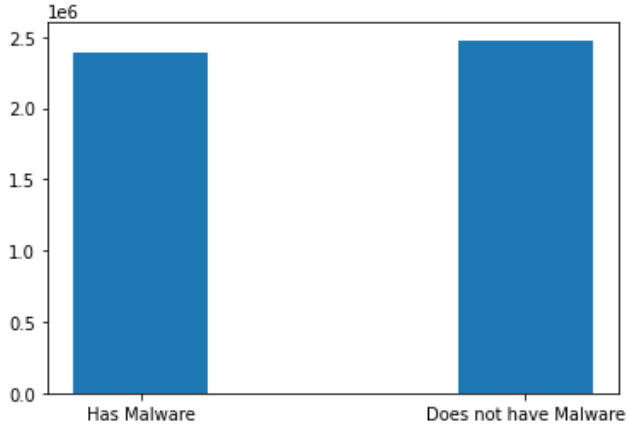

Fig. 1. Frequency of yes's and no's in the target variable of the dataset

III. EDA and Preprocessing

The approach taken was to do exploratory data analysis and preprocessing then save the modified dataset concluded from the inferences of the EDA and preprocessing steps. Initially for dealing with the NaN values the following steps were taken:

1) Dropping the columns which contained more than 30% of its entries as NaN values. Hence 'PuaMode' column was dropped.
2) For columns with lesser NaN or missing values, they were replaced with mean for numerical columns and with mode for the categorical columns.
3) The test dataset also contained missing values which were replaced with mean or mode of their respective columns in the train dataset.

Moving forward, initially to deal with the categorical columns we used one hot encoding. For using this approach the following columns were dropped due to excessive unique values:

1) 'AvSigVersion' with 8399 unique classes.
2) 'OsBuildLab' with 578 unique classes.
3) 'Census_OSVersion' with 337 unique classes.

Few other columns which had more than 20 unique classes of values were dropped. Outlier detection was done by checking the distribution with the help of violin plots of the values in the column as show in Figure:2 and Figure:3. The outlier values were dropped depending on the distribution. The final dataset after this was saved as a 'csv' file and used as

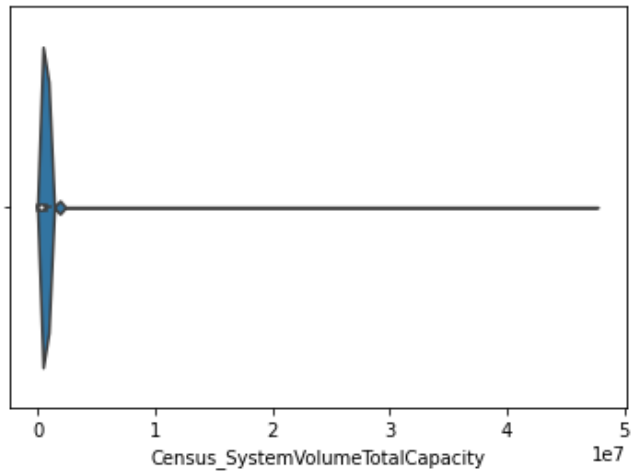

Fig. 2. Outliers in column '$Census\_SystemVolumeTotalCapacity$' in the dataset.

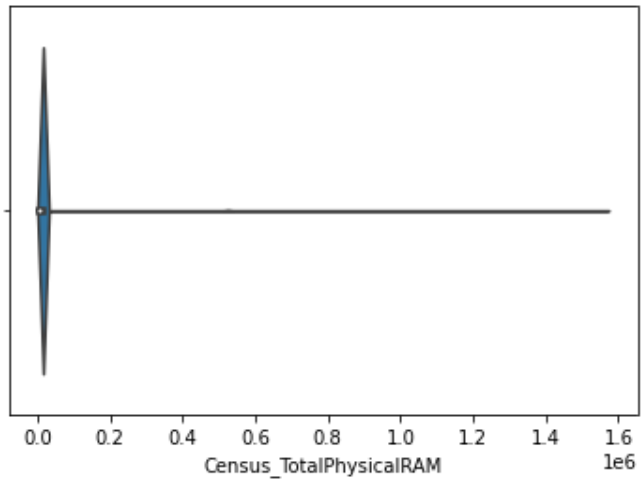

Fig. 3. Outliers in column '$Census\_TotalPhysicalRAM'$ in the dataset.

the training dataset in further notebooks for training. Categorical columns(EngineVersion, AppVersion, AvSigVersion, OsBuildLab, Census_OSVersion, Census_OSBranch, Census_OSEdition, Census_OSSkuName) were also handled by using a label encoder to encode the unique classes of values in a few models. Few columns in the test dataset had new class of unique values which were not present in the train datset, the columns obtained by one-hot encoding these were dropped for the model computation.
Normalization was applied to the dataset before all the models were trained, both min-max and standard normalization procedures were applied in different models.

IV. Model Selection and Training Process

Initially we used pandas to handle the data, wherein to reduce the ram consumption we checked the range of values in a column and downsized the number of bytes used to store the values in the column. This resulted in reduced ram usage, which helped us in training the Naive-Bayes models. We have tried many different models on a couple of different version of processed train dataset. As the dataset that we are using has an extremely large number of rows and columns we have decided to use DASK-ML to load the datasets onto memory and also make the models. Using dask to load the train and test datasets significantly reduced the dataset size and allowed us load the entire dataset and also one-hot encode columns.

We tried various combinations of one-hot encoding and label encoding on different columns and tried to train the models on this data. The best dataset was obtained after one-hot encoding some categorical that we thought were useful and the final column count was about 280 columns with about 6 million rows.

As this is a very large amount of data, training a model on the whole dataset would take a lot of memory. So the solution we found was to use the Incremental wrapper of dask to train models incrementally. The incremental wrapper in dask-ml can be used to train models which have partial fit methods. This incremental training allows us to train the model on the dataset in batches thereby reducing the memory required. The different models that we have used for training are:

- MLPClassifier with different layer activation functions
- SGDClassifier with different loss funtions like log, hinge, modified hubert, etc.
- Bernoulli-NB
- Multinomial-NB
- Gaussian-NB
- Logistic Regression
- Xgboost
- Linear SVM

Using PCA to reduce the dimensionality of the dataset was also tried out but not followed through as it took very long time to execute(more than 9-hrs). All of these models were trained using incremental wrapper of dask-ml. Out of all of the models used MLPClassifier has given the greatest Kaggle Score with Standard Scaler for scaling the data. Other scalers like MinMaxScaler and RobustScaler gave a worse score. We used dask-ml for scaling the data as well.

We have tried all of these models on different dataset variation with label encoding, a mixture of label encoding and one-hot encoding etc. However the highest we were able to obtain was with MLPClassifier and we obtained a score of 0.62 with a lot of hyperparameter tuning. At the end two hidden layers and 'relu' activation in the MLPClassifier gave the best score.

V. Conclusion

We were able to apply all of the concepts we have learn in the Machine Learning course over the semester to solve this particular problem. This project has helped understand about a lot of Machine Learning tools that used to solve big data problems like dask, downsizing of datasets, managing system memory and also about incremental training.

VI. Acknowledgement

We would like Prof. Dinesh and Prof. Neelam for teaching the Machine Learning course in a easy to understand and compact method which has helped us a lot in grasping the Machine Learning concepts. We would also like to thank all of the TAs for their work in helping us understand the practical side of Machine Learning.

VII. References

- https://ml.dask.org/
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- https://scikit-learn.org/0.15/modules/generated/sklearn.linear_model.SGDClassifier.htmlsklearn.linear_model.SGDClassifier
- https://ml.dask.org/incremental.html