ASSIGNMENT 2
Team Name: JAM

Mrinal M
IMT2018044
International Institute of Information Technology, Bangalore
Mrinal.M@iiitb.org

Kanumuru Shree Jhanwwee Reddy
IMT2018035
International Institute of Information Technology, Bangalore
Shree.Jhanwwee@iiitb.org

Vattikuti Sai Anvith
IMT2018528
International Institute of Information Technology, Bangalore
Sai.Anvith@iiitb.org

***Abstract*—The goal of this assignment is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine.**

Each row in this data-set corresponds to a machine, uniquely identified by a 'MachineIdentifier'. 'HasDetections' is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, you must predict the value for 'HasDetections' for each machine in test.csv.

I. Introduction

The malware industry continues to be a well-organized, well-funded market dedicated to evading traditional security measures. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways. Can you help protect more than one billion machines from damage BEFORE it happens?

So as discussed we have to study the data set, understand it and be able to predict if malware is present. In this assignment report, We shall briefly note our ideas, implementation and observations.

Firstly on this report we shall take through the data set given then the encoding we used for different columns. Next we shall discuss the different models tried and how we finally decided to balance the given data set.

II. Dataset

The dataset used in the project is from the IIITB ML: Malware Detection competition which was hosted on Kaggle.

Here we have 2 files: Malwaretest.csv and Malwaretrain.csv. One is a training dataset and the other is to be used for testing. Another file: Malware sample submission.csv is given to let us know how the output is supposed to be like. It contains a Machine Identifier column and a Has Detection output column. The Machine Identifier column holds the unique ID.

In both train dataset has 83 columns and the test dataset has 82 columns. We usually do not use all the features to compute and predict the output. Some features might have excessively high NaN values or be heavily skewed. Some might not even have any impact on the prediction result. While many others need to be encoded in different ways so that they could be used for prediction. Such things we tried to address firstly.

III. Feature Engineering

We tried to make the dataset more manageable by first removing the columns with too many NaN values.

The columns we removed are - CensusInternalBatteryType','DefaultBrowsersIdentifier','PuaMode', 'CensusProcessorClass', 'CensusIsWIMBootEnabled', 'IsBeta', 'CensusIsFlightsDisabled', 'CensusIsFlightingInternal', 'AutoSampleOptIn', 'CensusThresholdOptIn', 'SMode', 'CensusIsPortableOperatingSystem', 'CensusDeviceFamily', 'UacLuaenable', 'CensusIsVirtualDevice', 'Platform', 'CensusOSSkuName', 'CensusOSInstallLanguageIdentifier', 'Processor'

A. Encoding

For the other non integral features, we used different encoding methods. Primarily One-hot encoding and Label encoding techniques.

If a feature has more than 50 unique values, we have decided to choose label encoding. For those features with less than 50 unique values, We did one hot encoding.

B. Label Encoding

The following features have more than 50 unique values- 'MachineIdentifier', 'EngineVersion', 'AppVersion', 'AvSigVersion', 'OsBuildLab', 'CensusOSVersion'. Hence we used Label Encoding for these columns.

C. One Hot Encoding

Similarly for the others which have less than 50 unique values, namely - 'ProductName', 'OsVer', 'OsPlatformSubRelease', 'SkuEdition', 'SmartScreen', 'CensusMDC2FormFactor', 'CensusPrimaryDiskTypeName' , 'CensusPowerPlatformRoleName', 'CensusOSArchitecture', 'CensusFlightRing', 'CensusActivationChannel', 'CensusGenuineStateName', 'CensusOSWUAutoUpdateOptionsName', 'CensusOSInstallTypeName', 'CensusOSEdition', 'CensusOSBranch', 'CensusChassisTypeName', we used One Hot encoding.

D. Removing NaNs

To remove NaN values from the rest, we used SimpleImputer function from sklearn.impute library. Through this we replaced NaN values with median for each column.

CODE SNIPPETS

Below are code snippets showing how we implemented the encoding schemes in general.

```
label = []
for col in train.columns:
    if train[col].dtypes == object:
        unique = len(train[col].unique())
        if unique>50:
            label.append(col)
        print("Feature {} has {} unique categories".format(col,unique))
print(label)
def handle_non_numerical_data(df,columns):
#     columns = df.columns.values
    for column in columns:
        text_digit_vals = {}
        def convert_to_int(val):
            return text_digit_vals[val]

        if df[column].dtype != np.int64 and df[column].dtype != np.float64:
            column_contents = df[column].values.tolist()
            unique_elements = set(column_contents)
            x = 0
            for unique in unique_elements:
                if unique not in text_digit_vals:
                    text_digit_vals[unique] = x
                    x+=1

            df[column] = list(map(convert_to_int, df[column]))

    return df
```

Fig. 1. Label Encoding

```
SkuEdition = []
for i in train['SkuEdition']:
    if i in home:
        SkuEdition.append(0)
    elif i in pro:
        SkuEdition.append(1)
    elif i in other:
        SkuEdition.append(2)
train['SkuEdition'] = SkuEdition
SkuEdition = []
for i in test['SkuEdition']:
    if i in home:
        SkuEdition.append(0)
    elif i in pro:
        SkuEdition.append(1)
    elif i in other:
        SkuEdition.append(2)
test['SkuEdition'] = SkuEdition
dummies = pd.get_dummies(train['SkuEdition'], prefix='SkuEdition',dummy_na=False)
train = train.drop('SkuEdition',1)
train = pd.concat([train,dummies],axis=1)
dummies = pd.get_dummies(test['SkuEdition'], prefix='SkuEdition',dummy_na=False)
test = test.drop('SkuEdition',1)
test = pd.concat([test,dummies],axis=1)
```

Fig. 2. One Hot Encoding

As seen from above figures, for label encoding we assigned each unique element with a number if the element is not a numeric value already.

For One Hot encoding we took a specific feature - 'SkuEdition' as an example. Home, Pro contain the unique element - 'Home', 'Pro' respectively. In other we have included all the less frequently noted unique elements. We now encoded it into One-Hot fashion by considering 3 new values - 0, 1, 2.

Similarly we have repeated the same for other columns too based on the number of unique elements they have.

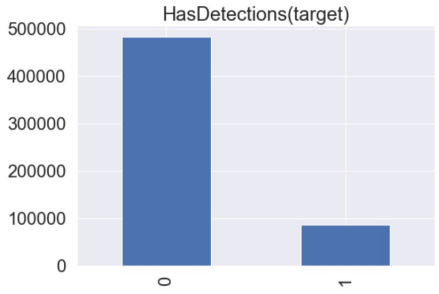

Fig. 3. Count vs HasDetection unique values

IV. APPROACH TO IMBALANCED DATA

The current data set we have is evidently(from the above plot), imbalanced. This is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. They tend to predict the majority class for everything because they are usually designed to improve accuracy by reducing the error. Thus, they do not take into account the class distribution / proportion or balance of classes.

Dealing with imbalanced datasets entails strategies such as balancing classes in the training data (data preprocessing) before providing the data as input to the machine learning algorithm. The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done in order to obtain approximately the same number of instances for both the classes.

A. Random Under-Sampling

Random Undersampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

1) Advantages: It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

2) Disadvantages: It can discard potentially useful information which could be important for building rule classifiers.

The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representative of the population. Thereby, resulting in inaccurate results with the actual test data set.

B. Random Over-Sampling

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

1) Advantages: Unlike under sampling this method leads to no information loss.

Outperforms under sampling

2) Disadvantages: It increases the likelihood of overfitting since it replicates the minority class events.

Although doing the above two methods help overcome the imbalanced nature of the dataset and prevented the model from predicting just the majority class, when inbuilt parameters of models for imbalanced data were used, the model performed better.

It is better not to use accuracy as a metric. So we used other metrics such as AUC score, confusion matrix, area under curve, precision, recall, f1-score, etc. These helped us judge the performance of the model better.

V. Models

Random forest, decision tree, logistic regression ,weighted logistic regression are the models we initially tried. After that tried the model with different sampling methods.

We experimented with different models, starting from logistic regression, weighted logistic regression. Both the models failed in indentifying the minority class and the recall score was around 0.10 for the minority class. After then we tried DecisionTree Model and Random Forest model, both of which was overfitting the data. We tried tweeking the hyperparameters(mainly the maxdepth), but when the model was not overfitting it was performing poorly on test set. After failing with these models we went to on to try for complex models like xgboost and lgbm. Lgbm model was performing relatively well on both the train and validation set, and it gave a decent score on the test set as well(.69).

Now we tried to balance the data using imblearn in sklearn, but all attempts resulted in a worse model. We tried SMOTE and RandomOverSampling as well as RandomUndersampling. Then we used the inbuilt isunbalance parameter to balance the data and that worked the best and this model gave us the best score of(.7). We tried tweaking the parameters of the lgbm model(like maxdepth, numleaves etc), but without any success.

The sampling methods we tried are - Smote, tomek, random under sampling and random over sampling. We did not get desired results. All the models were over-fitted.

So shifted to xgboost and lgbm. The observation we noted here is - Lgbm was running faster and was little better than xgboost. However, sampling with lgbm was making the model worse. After that we used inbuilt parameter - IsUnbalance: True - to sample and that worked the best.

The result scores we observed are -

1) Lgbm plus hyperparameter tuning with inbuilt function isunbalanced-0.703077
2) Xgboost (submitted based on AUC score)- 0.69581
3) Xgboost with oversampling- 0.60607
4) Xgboost-0.50085
5) Random forest with undersampling- 0.49856
6) Decision tree with oversampling- 0.49147

Acknowledgment

This Assignment has been a wonderful opportunity for us to try and experiment around with different models. We also got to try practically on how to deal with imbalanced data. We would like to take this opportunity to thank Prof. Raghavan, Prof. Neelam Sinha and the TAs for the encouragement.

References

1) https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/
2) https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e: :text=Feature