DDOS ATTACK IDENTIFICATION

AI-511 Machine Learning

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY BANGALORE

16th December, 2022

Team Name :- WHIZZ
Team Members:- Anisha Rani(MT2022153)
Kshitija Shah(MT2022147)

Under guidance of:- Harshita Soni

Problem Statement

- We are given various features regarding events in a network. We need to identify if those events were related to a DDoS attack, or was the network intrusion benign.

- This is a **binary classification** problem involving approximately 80 features and 16 million training rows.

- The metric to be used is the **F1 score**. The dataset is imbalanced.

Dataset Overview

- Dataframe head overview using dask library.

	index	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	...	Fwd Seg Size Min	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	0	53.0	17.0	2291.0	1.0	1.0	49.0	164.0	49.0	49.0	...	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	1	80.0	6.0	5785762.0	3.0	1.0	0.0	0.0	0.0	0.0	...	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	2	80.0	6.0	13513.0	3.0	4.0	287.0	935.0	287.0	0.0	...	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
3	3	80.0	6.0	1610.0	3.0	4.0	308.0	935.0	308.0	0.0	...	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	4	53.0	17.0	1115.0	1.0	1.0	36.0	73.0	36.0	36.0	...	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 80 columns

1.Preprocessing Overview:

- Analysing the size of the training data provided.
- Analysing columns and data type.(79 columns :float64 ,Label:int64)
- Dropping columns with single unique values.
- Handling columns with infinite values.
- Understanding data distribution.(Imbalance data)
- Finding Correlation and reducing the feature set.

1.1 Observing and Cleaning the DATA

- Check For NULL values for the features provided.
- Check for special characters (missing values) in the data.
- Observing the unique value counts for each feature.
- 8 columns with single unique value is dropped.
- Dropped index.
- Dropped the columns with single unique value.
- ['index','Bwd URG Flags' , 'Fwd Byts/b Avg' , 'Fwd Pkts/b Avg','Fwd Blk Rate Avg','Bwd Byts/b Avg',' Bwd Pkts/b Avg','Bwd Blk Rate Avg','Bwd PSH Flags'] using nunique().compute().

- Checking the distribution of data using describe().

- Replacing 'inf' with 'nan'(2 columns-flowbytes/s,flowpack/s) discarding those rows,as the distribution of those rows are similar to the label distribution considering the fact that the data is imbalanced.
-
- Raw data: Features 80 features after processing-71
- rows:15173222 rows: 15139506

1.2 SIZE Optimization:

- Observed the data type for each feature and compared with the min and max range of type the column falls in. Reduced the data type of each column accordingly.

  - Initial data size: 7.45GB
  - Reduced data size: 5.77GB

```python
def reduce_mem_usage(df):

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min().compute()
            c_max = df[col].max().compute()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')



    return df
```

1.3 Correlation and data type Conversion:

- Removed those column which are highly correlation (corr > 0.9).
- 31 columns filtered out from the upper triangle of correlation matrix.

```
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.9)]
print(); print(to_drop)
len(to_drop)
```

```
['TotLenBwdPkts', 'FwdPktLenStd', 'BwdPktLenStd', 'FwdIATTot', 'FwdIATMean', 'FwdIATStd', 'FwdIATMax', 'FwdIATMin', 'FwdHeaderLe
n', 'BwdHeaderLen', 'FwdPkts/s', 'PktLenMin', 'PktLenMax', 'PktLenMean', 'PktLenStd', 'SYNFlagCnt', 'CWEFlagCount', 'ECEFlagCn
t', 'PktSizeAvg', 'FwdSegSizeAvg', 'BwdSegSizeAvg', 'SubflowFwdPkts', 'SubflowFwdByts', 'SubflowBwdPkts', 'SubflowBwdByts', 'Fwd
ActDataPkts', 'ActiveMax', 'ActiveMin', 'IdleMean', 'IdleStd', 'IdleMax']
31
```

+ Code + Markdown

```
train2 = d.drop(d[to_drop], axis=1)
```

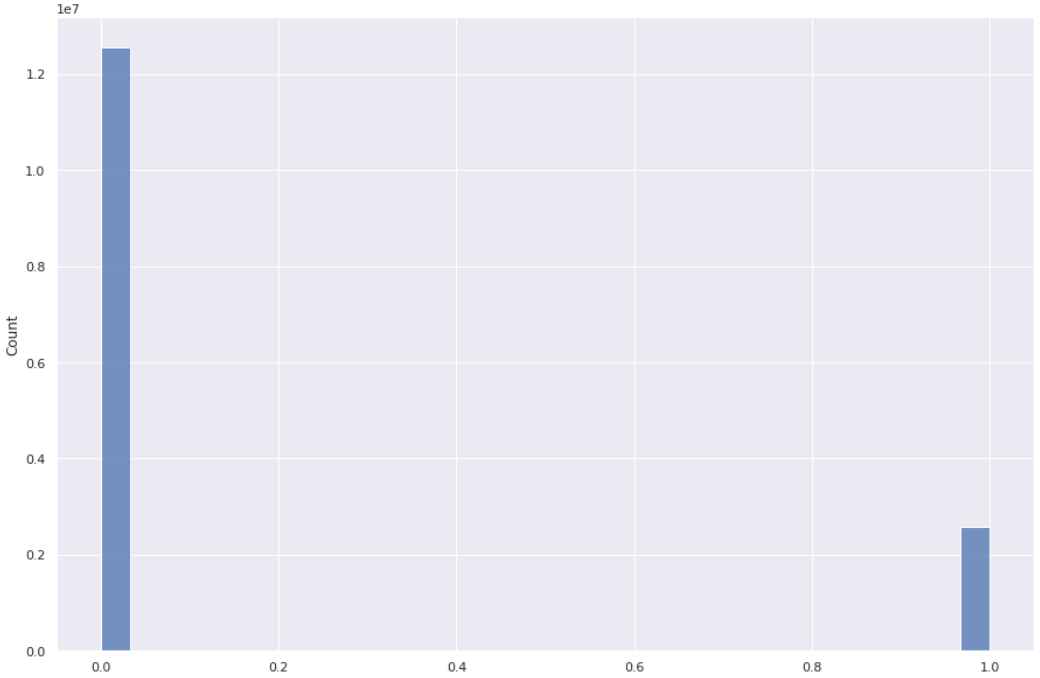
- Further analysed columns with 2 unique values(i.e 0,1)INT (9 column)
- Converted FLOAT to INT8

```
train2['Protocol'] = train2['Protocol'].astype(np.int8)
train2['FwdURGFlags'] = train2['FwdURGFlags'].astype(np.int8)
train2['FwdPSHFlags'] = train2['FwdPSHFlags'].astype(np.int8)
train2['FINFlagCnt'] = train2['FINFlagCnt'].astype(np.int8)
train2['URGFlagCnt'] = train2['URGFlagCnt'].astype(np.int8)
train2['ACKFlagCnt'] = train2['ACKFlagCnt'].astype(np.int8)
train2['PSHFlagCnt'] = train2['PSHFlagCnt'].astype(np.int8)
train2['RSTFlagCnt'] = train2['RSTFlagCnt'].astype(np.int8)
train2['Label'] = train2['Label'].astype(np.int8)
```

- SIZE:
- Before:5.77GB After:-3.06GB

1.4 IMBALANCE DATA CHECK

- HIST PLOT FOR LABEL
-



1.5 Dimentionality Reduction

To determine the importance of features over the predicted outcome we have removed the irrelevant variables and current dataset contains 40 features. Here, in our case we need to identify the impact of a variable over the Label column. Ideally, the variables with less importance can be ignored in model building. After splitting the training data to Xtrain and Ytrain(Label) we tried to optimize the data further :-

Experiments Conducted And Challenges Faced

- We applied PCA to the training and testing dataset in order to reduce the dimension of features to 25 .
- But due to fit_transform ,although the size of features reduce ,we did not see any significant reduction in file size.
- Pandas still could’nt read the entire Xtrain due to increase in size of large dataset after standardization.

Apart from some hardware constraints due to ram overflow(as the dataset was large), the project itself was fun and we learned a lot from it.
Further we applied different models and observed the F1 scores along with hyperparameter tuning.

2. Training the model

2.1 Logistic Regression:

Logistic regression is a classification technique borrowed by machine learning from the field of statistics. Logistic Regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determines an outcome. The intention behind using logistic regression is to find the best fitting model to describe the relationship between the dependent and the independent variable. For our dataset we have applied logistic regression initially with the default parameters and again with hyperparameter tuning. The F1 score was not convincing enough hence we moved on to other models.(as the dataset is highly imbalanced).

MODEL Accuracy : 0.8792837758378642
TEST DATA Accuracy:0.48205

HYPER parameter tuning

-GRID SEARCH CV
-RANDOMIZED SEARCH CV

Grid search –goes through all intermediate combination of hyperparameters.
Randomized goes through fixed no. of hyperparameters setting.

2.1.1 After applying randomized search cv

```
# from sklearn.linear_model import LogisticRegression
# classifier = LogisticRegression(random_state = 1)
# classifier.fit(Xtrain, Ytrain)
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import GaussianNB

params = {'C': [ 0.01, 0.1, 10, 100] }
x_clf_1 = LogisticRegression(n_jobs=-1)

random_clf_1 = RandomizedSearchCV(x_clf_1, param_distributions=params, scoring='f1', verbose=10, cv=2)
random_clf_1.fit(Xtrain,Ytrain)
```

MODEL ACCURACY:
'C': 0.01
SCORE:0.44976703897311565
TEST DATA ACCURACY:0.39153

2.2 Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Hyperparameter Tuning:

```
%%time
from scipy.stats import randint

params = {'max_depth' : [3,5,8,None],
          'max_features': ['auto','log2',None],
          }
x_clf_5 = DecisionTreeClassifier()

random_clf_5 = RandomizedSearchCV(x_clf_5, param_distributions=params, scoring='f1', verbose=1, cv=2, n_jobs=3)
random_clf_5.fit(Xtrain,Ytrain)
```

Best parameters and scores:

Hyperparameters output:
{'max_features': 'auto', 'max_depth': None}
MODEL f1 SCORE:
0.956016453545824 4
TEST DATA SCORE:
0.95586

```
print(random_clf_5.best_params_)
print(random_clf_5.best_score_)
```

```
{'max_features': 'auto', 'max_depth': None}
0.9560164535458244
```

2.3 Random Forest

- Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

```
model3 = RandomForestClassifier(n_estimators=500, max_depth=40,
                                random_state=42, n_jobs=-1, verbose=1)
model3.fit(Xtrain,Ytrain)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 20 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 160 tasks      | elapsed: 38.0min
[Parallel(n_jobs=-1)]: Done 410 tasks      | elapsed: 91.3min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 109.6min finished
```

```
RandomForestClassifier(max_depth=40, n_estimators=500, n_jobs=-1,
                       random_state=42, verbose=1)
```

- TEST DATA SCORE : 0.9564

2.3.1 Random Forest with Hyperparameter Tuning

```
params={'n_estimators' : [100,200,300,500,800],
        'max_depth' : [10,20,40,60,80]
        }

x_clf_5 = RandomForestClassifier()

random_clf_5 = RandomizedSearchCV(x_clf_5, param_distributions=params, scoring='f1', verbose=1, cv=2, n_jobs=3)
random_clf_5.fit(Xtrain,Ytrain)
```

- TEST DATA SCORE : 0.96539

2.4 XGBOOST

- XGBoost is an implementation of Gradient Boosted decision trees. Decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model.

```
from xgboost import XGBClassifier
model4 = XGBClassifier(max_depth=5, subsample=1.0,
                       colsample_bytree=0.6, min_child_weight=1,
                       gamma=1,n_jobs=-1)
model4.fit(Xtrain,Ytrain, verbose=10)
```

- MODEL f1 SCORE: 0.969987
- TEST DATA SCORE : 0.96963

2.4.1 XGBOOST with Hyperparameter Tuning

```
model6 = XGBClassifier(max_depth=random_clf_6.best_params_['max_depth'], subsample=random_clf_6.best_params_['subsam
                       colsample_bytree=random_clf_6.best_params_['colsample_bytree'], min_child_weight=random_clf_6
                       gamma=random_clf_6.best_params_['gamma'],n_jobs=-1)
model6.fit(Xtrain,Ytrain, verbose=10)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.6,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=5, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=5, max_leaves=0, min_child_weight=10,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=-1, num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

- MODEL f1 SCORE : 0.97001
- TEST SCORE : 0.9697

Stochastic Gradient Descent

- Stochastic gradient descent considers only 1 random point while changing weights unlike gradient descent which considers the whole training data. As such stochastic gradient descent is much faster than gradient descent when dealing with large data sets.

- MODEL SCORE : 0.84067
- TEST SCORE : 0.42185

Challenges Faced

- Could not implement ADABOOST , as it does not have system optimizations.
- AdaBoost is sensitive to noise data. It is highly affected by outliers because it tries to fit each point perfectly.
- AdaBoost is slower compared to XGBoost.

References

https://medium.com/codex/do-i-need-to-tune-logistic-regression-hyperparameters-1cb2b81fca69

https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

Train-Test Data after preprocessing

https://www.kaggle.com/datasets/kshitijaashah/traintest?select=X_train.csv

https://www.kaggle.com/datasets/kshitijaashah/traintest?select=Y_train.csv

https://www.kaggle.com/datasets/kshitijaashah/traintest?select=X_test.csv

THANK YOU