10/23/2020

Information Security Analysis and Audit

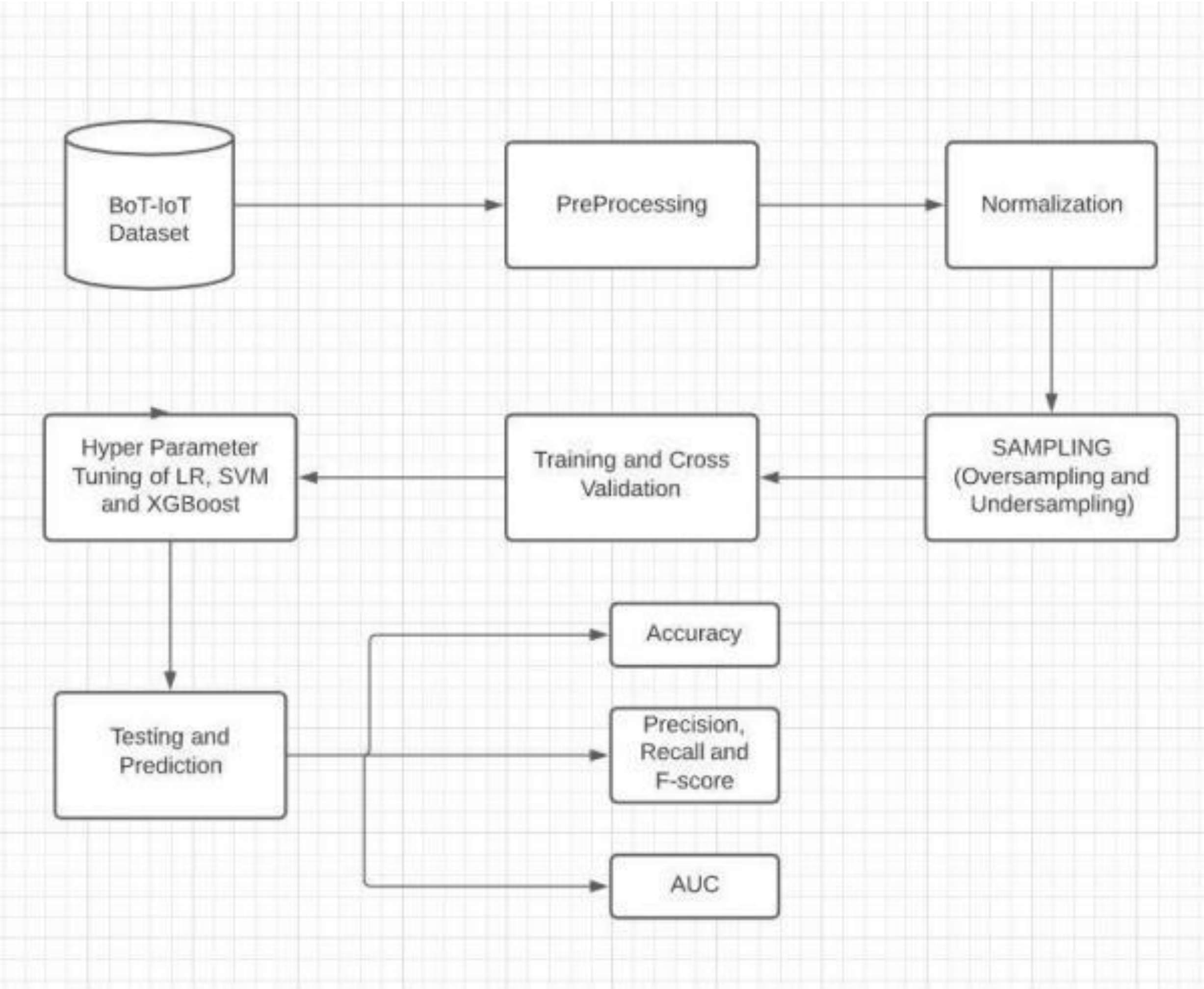
Project Review 2

Intrusion Detection System using Machine Learning (XGBoost Algorithm)

Kushagra Agarwal
18BIT0231

Intrusion Detection System using XGBoost

Design and Description of system



Dataset

The BoT-ioT dataset was created by designing a realistic network environment in the Cyber Range Lab of the center of UNSW Canberra Cyber. The dataset has 10 best features and is split into training dataset and testing dataset. The environment incorporates a combination of normal and botnet traffic. The dataset's source files are provided in different formats, including the original pcap files, the generated argus files and csv files.

Dataset link:
https://cloudstor.aarnet.edu.au/plus/s/umT99TnxvbpkkoE?path=%2FCSV%2FTraning%20and%20Testing%20Tets%20(5%25%20of%20the%20entier%20dataset)%2F10-best%20features%2F10-best%20Training-Testing%20split

In [3]:
```
data_train = pd.read_csv('UNSW_Training.csv')
data_test = pd.read_csv('UNSW_Testing.csv')
```

In [4]:
```
data_train.head()
```

Out[4]:

	pkSeqID	proto	saddr	sport	daddr	dport	seq	stddev	N_IN_Conn_P_SrcIP	min	state_number	mean	N_IN_Conn_P_DstIP
0	3142762	udp	192.168.100.150	6551	192.168.100.3	80	251984	1.900363	100	0.000000	4	2.687519	100
1	2432264	tcp	192.168.100.150	5532	192.168.100.3	80	256724	0.078003	38	3.856930	3	3.934927	100
2	1976315	tcp	192.168.100.147	27165	192.168.100.3	80	62921	0.268666	100	2.974100	3	3.341429	100
3	1240757	udp	192.168.100.150	48719	192.168.100.3	80	99168	1.823185	63	0.000000	4	3.222832	63
4	3257991	udp	192.168.100.147	22461	192.168.100.3	80	105063	0.822418	100	2.979995	4	3.983222	100

In [4]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2934817 entries, 0 to 2934816
Data columns (total 19 columns):
 #   Column             Dtype
---  ------             -----
 0   pkSeqID            int64
 1   proto              object
 2   saddr              object
 3   sport              object
 4   daddr              object
 5   dport              object
 6   seq                int64
 7   stddev             float64
 8   N_IN_Conn_P_SrcIP  int64
 9   min                float64
 10  state_number       int64
 11  mean               float64
 12  N_IN_Conn_P_DstIP  int64
 13  drate              float64
 14  srate              float64
 15  max                float64
 16  attack             int64
 17  category           object
 18  subcategory        object
dtypes: float64(6), int64(6), object(7)
memory usage: 425.4+ MB
```

In [5]:
```
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 733705 entries, 0 to 733704
Data columns (total 19 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   pkSeqID            733705 non-null  int64
 1   proto              733705 non-null  object
 2   saddr              733705 non-null  object
 3   sport              733705 non-null  object
 4   daddr              733705 non-null  object
 5   dport              733705 non-null  object
 6   seq                733705 non-null  int64
 7   stddev             733705 non-null  float64
 8   N_IN_Conn_P_SrcIP  733705 non-null  int64
 9   min                733705 non-null  float64
 10  state_number       733705 non-null  int64
 11  mean               733705 non-null  float64
 12  N_IN_Conn_P_DstIP  733705 non-null  int64
 13  drate              733705 non-null  float64
 14  srate              733705 non-null  float64
 15  max                733705 non-null  float64
 16  attack             733705 non-null  int64
 17  category           733705 non-null  object
 18  subcategory        733705 non-null  object
dtypes: float64(6), int64(6), object(7)
memory usage: 106.4+ MB
```

In [6]:
```
data_train.shape
```

Out[6]: (2934817, 19)

There are 19 columns in both training and testing dataset. The training dataset has 2934817 rows while the testing dataset has 733705 rows. Target column for the dataset is ‘category’ column.

Data Preprocessing:

Data processing is the method of processing raw data and making it suitable to fit into the machine learning model. This step is used to pre-process the data like the data should be cleaned and in single format. We have dropped some columns from out test and train dataset as they were unnecessary for our prediction. Preprocessing helps in better prediction of the model.

The object datatypes are converted to int or float. There are hexadecimal values in the destination port (dport) and source port (sport) so they are also changed in the decimal form during preprocessing. The theft category from our target variable is also dropped as it is very less in quantity.

In [11]:
```
#Dropping columns from both the dataset as they are not important to get our target values
data_train.drop(["pkSeqID","seq","subcategory"], axis=1, inplace=True)
data_test.drop(["pkSeqID","seq","subcategory"], axis=1, inplace=True)
```

In [12]:
```
data_train['category'].value_counts()
```

Out[12]:
```
DDoS              1541315
DoS               1320148
Reconnaissance      72919
Normal                370
Theft                  65
Name: category, dtype: int64
```

In [15]:
```
indexNames = data_train[data_train['category']=='Theft'].index
data_train.drop(indexNames , inplace=True)
```

In [16]:
```
indexNames = data_test[data_test['category']=='Theft'].index
data_test.drop(indexNames , inplace=True)
```

In [17]:
```
data_train['category'].value_counts()
```

Out[17]:
```
DDoS              1541315
DoS               1320148
Reconnaissance      72919
Normal                370
Name: category, dtype: int64
```

In [24]:
```
check='0x'
s_res = set([i for i in data_train['sport'] if i.startswith(check)])
s_res
```

Out[24]:
```
{'0x0008', '0x000d', '0x0011', '0x0303'}
```

Some of the values in source port (sport) are in Hexadecimal form, since they are low in number we will directly replace them with their corresponding Decimal form

In [25]:
```
data_train['sport']=data_train['sport'].replace(['0x0303'],'771')
data_train['sport']=data_train['sport'].replace(['0x0011'],'17')
data_train['sport']=data_train['sport'].replace(['0x000d'],'13')
data_train['sport']=data_train['sport'].replace(['0x0008'],'8')
```

In [26]:
```
data_test['sport']=data_test['sport'].replace(['0x0303'],'771')
data_test['sport']=data_test['sport'].replace(['0x0011'],'17')
data_test['sport']=data_test['sport'].replace(['0x000d'],'13')
data_test['sport']=data_test['sport'].replace(['0x0008'],'8')
```

In [27]:
```
#Converting object datatype to int datatype
data_train["sport"] = data_train["sport"].astype(str).astype(int)
data_test["sport"] = data_test["sport"].astype(str).astype(int)
```

In [30]:
```
data_train["dport"] = data_train["dport"].apply(lambda x: int(x,16) if len(x)>1 and x[1]=="x" else int(x))
data_test["dport"] = data_test["dport"].apply(lambda x: int(x,16) if len(x)>1 and x[1]=="x" else int(x))
```

Encoding:

Data Encoding is the process of converting object data type into either int or float data type so that the columns can be fit into the Machine Learning model. Various techniques are available for encoding such as Label encoding, One Hot encoding, Target encoding , etc. We have used Label encoding in our project as the other encoding techniques introduce a large number of columns for our dataset which may slow down the processing.

In [31]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_train["saddr_enc"]= le.fit_transform(data_train.saddr)
data_train["daddr_enc"]= le.fit_transform(data_train.daddr)
data_train["proto_enc"]= le.fit_transform(data_train.proto)
data_train["category_enc"]= le.fit_transform(data_train.category)
```

In [32]:
```
data_train.head()
```

Out[32]:

nn_P_SrcIP	min	state_number	mean	N_IN_Conn_P_DstIP	drate	srate	max	attack	category	saddr_enc	daddr_enc	proto_enc	category_enc
100	0.000000	4	2.687519	100	0.0	0.494549	4.031619	1	DDoS	4	15	4	0
38	3.856930	3	3.934927	100	0.0	0.256493	4.012924	1	DDoS	4	15	3	0
100	2.974100	3	3.341429	100	0.0	0.294880	3.609205	1	DDoS	1	15	3	0
63	0.000000	4	3.222832	63	0.0	0.461435	4.942302	1	DoS	4	15	4	1
100	2.979995	4	3.983222	100	0.0	1.002999	4.994452	1	DDoS	1	15	4	0

In [35]:
```
data_train['category_enc'].value_counts()
```

Out[35]:
```
0    1541315
1    1320148
3      72919
2        370
Name: category_enc, dtype: int64
```

0 - DDoS
1 - DoS
3 - Reconnaissance
2 - Normal

In [36]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2934752 entries, 0 to 2934816
Data columns (total 16 columns):
 #   Column             Dtype  
---  ------             -----  
 0   sport              int32  
 1   dport              int64  
 2   stddev             float64
 3   N_IN_Conn_P_SrcIP  int64  
 4   min                float64
 5   state_number       int64  
 6   mean               float64
 7   N_IN_Conn_P_DstIP  int64  
 8   drate              float64
 9   srate              float64
 10  max                float64
 11  attack             int64  
 12  saddr_enc          int32  
 13  daddr_enc          int32  
 14  proto_enc          int32  
 15  category_enc       int32  
dtypes: float64(6), int32(5), int64(5)
memory usage: 324.7 MB
```

Normalization

Normalization or scaling is a method used in the preparation of machine learning model. The goal is to convert numeric column values in the order to use the same scale, without distorting the differences in the range of values and loss of information. We have implemented normalization by standard scalar.

In [38]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
features = data_train.iloc[:,:-1]
cols=features.columns
scaled_features= scaler.fit_transform(features)
data_train= pd.DataFrame(scaled_features,columns=cols)
```

In [39]:
```
features = data_test.iloc[:,:-1]
cols=features.columns
scaled_features= scaler.fit_transform(features)
data_test= pd.DataFrame(scaled_features,columns=cols)
```

In [40]:
```
data_train.head()
```

Out[40]:

	sport	dport	stddev	N_IN_Conn_P_SrcIP	min	state_number	mean	N_IN_Conn_P_DstIP	drate	srate	max	attack	saddr_
0	-1.380796	-0.094028	1.260991	0.715435	-0.685661	0.729327	0.301113	0.415159	-0.00763	-0.003355	0.543986	0.011229	1.301
1	-1.434107	-0.094028	-1.006649	-1.826659	1.914132	-0.113083	1.122991	0.415159	-0.00763	-0.003659	0.533940	0.011229	1.301
2	-0.302346	-0.094028	-0.769399	0.715435	1.319054	-0.113083	0.731954	0.415159	-0.00763	-0.003610	0.316991	0.011229	-1.281
3	0.825282	-0.094028	1.164956	-0.801621	-0.685661	0.729327	0.653814	-1.621898	-0.00763	-0.003397	1.033365	0.011229	1.301
4	-0.548442	-0.094028	-0.080342	0.715435	1.323028	0.729327	1.154811	0.415159	-0.00763	-0.002707	1.061389	0.011229	-1.281

Sampling

Data sampling is a method used for selecting observations from the information about the population based on the statistics from a sample. For our dataset we have used undersampling and oversampling. Oversampling is done for the Normal category and under sampling is done for DDoS and DoS. All the values are brought to 72919 equals to the value of Reconnaissance.

In [44]:
```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
samp_strat= { 0:72919, 1:72919, 3:72919, 2:370}
random_under= RandomUnderSampler(sampling_strategy=samp_strat,random_state=1)
X_rus,y_rus = random_under.fit_resample(X_train,y_train)
```

In [45]:
```
y_rus.value_counts()
```

Out[45]:
```
3    72919
1    72919
0    72919
2      370
Name: category_enc, dtype: int64
```

In [46]:
```
from imblearn.over_sampling import RandomOverSampler
samp_strat= { 0:72919, 1:72919, 2:72919, 3:72919}
random_under= RandomOverSampler(sampling_strategy=samp_strat,random_state=1)
Xres,yres = random_under.fit_resample(X_rus,y_rus)
```

In [47]:
```
yres.value_counts()
```

Out[47]:
```
3    72919
2    72919
1    72919
0    72919
Name: category_enc, dtype: int64
```

Training

The training of the dataset is done for the train dataset using the XGBoost model.

In [49]:
```
from xgboost import XGBClassifier
model_1 = XGBClassifier(random_state = 42)
model_1.fit(Xres, yres)
pred_1 = model_1.predict(X_test)
score1 = model_1.score(X_test,y_test)
print("Accuracy of base model: ",score1)
```

```
Accuracy of base model:  0.6174602114514148
```

This is considered as base model for further tuning.

Cross-validation

It is a process which is used to evaluate the machine learning model using the resampling procedure on a given data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. Generally, it is known as k-fold cross-validation. We have implemented the KFold and Stratified KFold techniques and have used 10 splits. The score is calculated using the cross-validation score.

```
In [128]: # from sklearn.model_selection import KFold,StratifiedKFold,cross_val_score
          # def evaluate_model(model):

          #     KF=KFold(n_splits=10,shuffle=True,random_state=42)
          #     score1 = cross_val_score(model, Xres, yres, scoring='accuracy', cv=KF)

          #     SKF= StratifiedKFold(n_splits=10,shuffle=True)
          #     score2 = cross_val_score(model, Xres, yres, scoring='accuracy', cv=SKF)

          #     list_scores=[np.mean(score1),np.mean(score2)]

          #     return list_scores
```

```
In [130]: # names=["KFold","Stratified KFold"]
          # print("KFOLD CROSS VALIDATION SCORES")
          # print("(For Base Model 0)")
          # scores_k1 = evaluate_model(model_1)

          # for (i,j) in zip(scores_k1,names):
          #     print(j,"-",round(i*100,2),'%')
```

Due to some error the code for K-fold did not give correct results, I will try to fix that and show the result in Review 3. However, during the hyperparameter tuning, model has used cross validation to achieve more accuracy.

Hyper Parameter Tuning

Choosing a set of optimal hyper parameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. Different parameters are used to get the accuracy of the model. For XGBoost, I have used the parameter learning rate, max_depth, n_estimators and subsample. I have given a range of values for each of the parameter and ran multiple models to get the best parameters for high accuracy. During the execution of different test models, I have also used 2 different parameters for some models, those are:

N_jobs = -1: This parameter increases the speed of training model as it makes use of all the cores of CPU for processing.

cv = 3,5: This parameter is for determining the folds of cross validation of model.

```
In [50]: from sklearn.model_selection import GridSearchCV
```

```
In [52]:  params={'max_depth':[5], 'n_estimators': [200,300,400],
                'learning_rate': [0.15,0.20], 'subsample':[0.3,0.5,0.7]}
```

```
In [53]: model_2 = GridSearchCV(XGBClassifier(),params,refit=True,verbose=3)
```

In [87]:
```
params_6 = {'max_depth':[5], 'n_estimators': [200,300,500],
            'learning_rate': [0.20,0.30], 'subsample':[0.5,0.6,0.7]}
model_6 = GridSearchCV(XGBClassifier(),params_4,refit=True,verbose=3,
                       cv=5,n_jobs=-1)
model_6.fit(Xres,yres)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 16.3min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed: 109.4min finished
```

Out[87]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, subsample=None,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=None),
             n_jobs=-1,
             param_grid={'learning_rate': [0.2, 0.3], 'max_depth': [5],
                         'n_estimators': [200, 300, 500],
                         'subsample': [0.5, 0.6, 0.7]},
             verbose=3)
```

In [88]:
```
model_6.best_params_
```

Out[88]:
```
{'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 500, 'subsample': 0.6}
```

In [89]:
```
model_6.best_estimator_
```

Out[89]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=500, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=0.6,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [91]:
```
pred_6 = model_6.predict(X_test)
```

In [92]:
```
score_6 = model_6.score(X_test,y_test)
print("Accuracy of 5th model: ",score_6*100)
```

```
Accuracy of 5th model:  90.48741227574007
```

Results

In [117]:
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score, f1_score, auc, roc_curve, roc_auc_score
from sklearn import metrics
```

In [121]:
```
print("Results of various Parameter tuned models ")
print("Base model 0: ",round(score1*100,2),"%")
print("Test model 1: ",round(score_2*100,2),"%")
print("Test model 2: ",round(score_3*100,2),"%")
print("Test model 3: ",round(score_4*100,2),"%")
print("Test model 4: ",round(score_5*100,2),"%")
print("Test model 5: ",round(score_6*100,2),"%")
```

```
Results of various Parameter tuned models
Base model 0:  61.75 %
Test model 1:  81.89 %
Test model 2:  72.12 %
Test model 3:  67.01 %
Test model 4:  81.89 %
Test model 5:  90.49 %
```

Confusion matrix for tuned model:

In [124]:
```
print(confusion_matrix(y_test,pred_1))
```

```
[[381229    647      0   3433]
 [   457  53532      0 276123]
 [     0      0    107      0]
 [     5      1      0  18157]]
```

Classification Report:

```
In [120]: # Classification Report of base model
          print(classification_report(y_test,pred_1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99    385309
           1       0.99      0.16      0.28    330112
           2       1.00      1.00      1.00       107
           3       0.06      1.00      0.11     18163

    accuracy                           0.62    733691
   macro avg       0.76      0.79      0.60    733691
weighted avg       0.97      0.62      0.65    733691
```

Inferences from the report:

- Recall of the model is 0.62, i.e. 62% of the predicted vaues were true(1).
- Precision of the model is 0.97, i.e. 97% of the predicted true values were correct.
- F1 score is not very high, 0.65 this is due to low recall.

Out of the three models we used i.e. Logistic regression, XG Boost and SVM, SVM performed best after hyper parameter tuning. Accuracy of the three models are

Logistic Regression: 96.63%

SVM: 99.73%

XGBoost: 90.49%

Complete Code

The complete code is uploaded on the GitHub.

GitHub link: https://github.com/PseudoKush/Intrusion-Detection-System/blob/main/Intrusion%20Detection%20using%20XGBoost.ipynb

Digital Signature