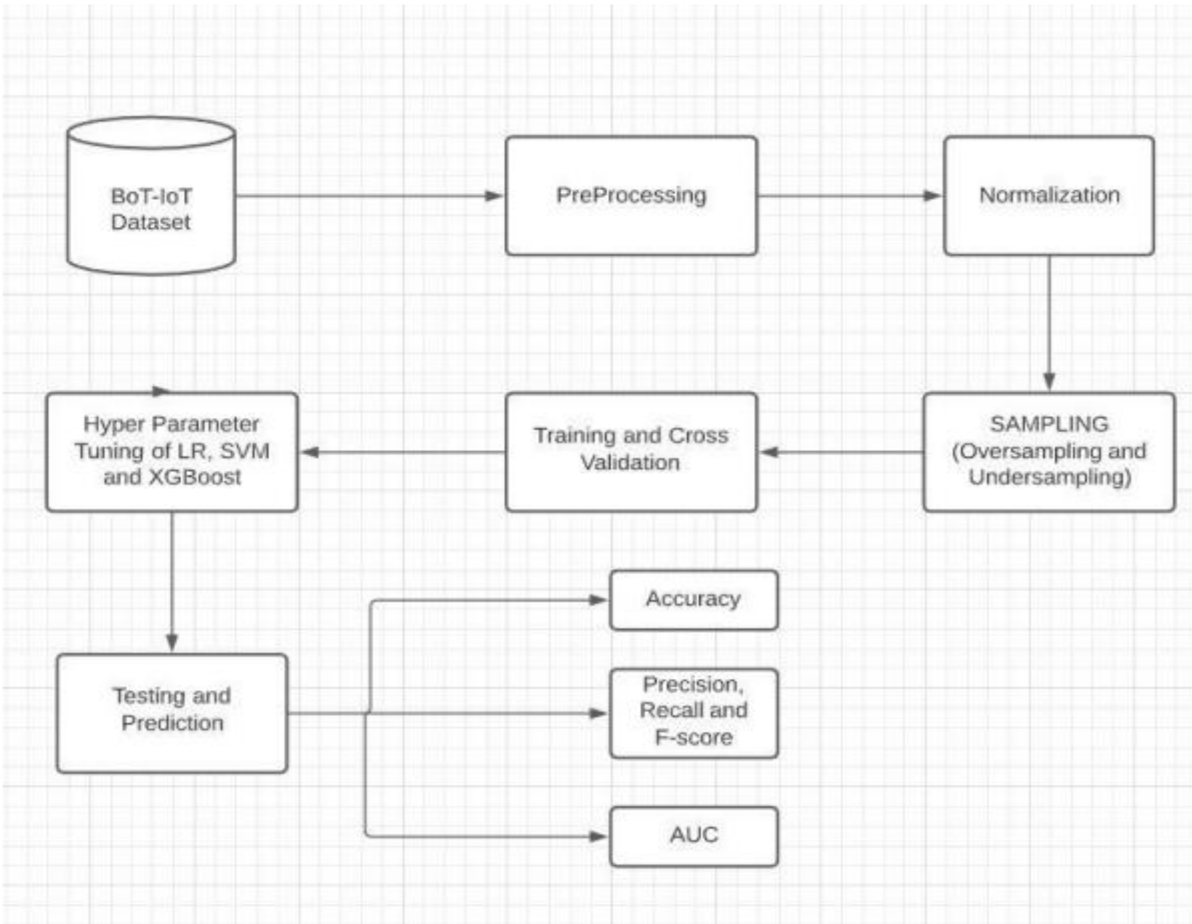
Intrusion Detection System using SVM

REVIEW 2

ROHAN JAIN
18BIT0429 | G2

Intrusion Detection System using SVM

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.

In [2]:
```
train = pd.read_csv("10-best features/10-best Training-Testing split/UNSW_2018_IoT_Botnet_Final_10_best_Training.csv")
test = pd.read_csv("10-best features/10-best Training-Testing split/UNSW_2018_IoT_Botnet_Final_10_best_Testing.csv")
```

In [3]:
```
train.head()
```

Out[3]:

	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]:
```
test.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	792371	udp	192.168.100.150	48516	192.168.100.3	80	175094	0.226784	100	4.100436	4	4.457383	100
1	2056418	tcp	192.168.100.148	22267	192.168.100.3	80	143024	0.451998	100	3.439257	1	3.806172	100
2	2795650	udp	192.168.100.149	28629	192.168.100.3	80	167033	1.931553	73	0.000000	4	2.731204	100
3	2118009	tcp	192.168.100.148	42142	192.168.100.3	80	204615	0.428798	56	3.271411	1	3.626428	100
4	303688	tcp	192.168.100.149	1645	192.168.100.5	80	40058	2.058381	100	0.000000	3	1.188407	100

There are 19 columns in both train and test dataset. The train dataset has 2934817 rows while the test dataset has 733705 rows

In [7]:
```
train.shape
```

Out[7]: (2934817, 19)

In [8]:
```
test.shape
```

Out[8]: (733705, 19)

Preprocessing

Data processing is the method of processing raw data and making it suitable for the machine learning model. This step is used to preprocess the data like the data should be cleaned and should be in one format. We have dropped some columns from out test and train dataset as they were not necessary for out prediction. Preprocessing help in better prediction of the model. Also, the object datatypes are converted to int or float. There are hexadecimal values in the dport and sport so there are also changed in the decimal form in preprocessing. The theft category from our target variable is also dropped as it is very less in quantity.

In [11]:
```
train.drop(["pkSeqID","seq","subcategory"], axis=1, inplace=True)
```

In [12]:
```
test.drop(["pkSeqID","seq","subcategory"], axis=1, inplace=True)
```

In [21]:
```
drop_theft = train[train['category']=='Theft'].index
train.drop(drop_theft , inplace=True)
```

In [22]:
```
drop_theft = test[test['category']=='Theft'].index
test.drop(drop_theft , inplace=True)
```

In [36]:
```
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
```

All Object datatypes are converted to Int and are appended at the end of dataset.

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

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

The category variable is encoded as category_enc where

0 – DdoS

1 – Dos

2 – Normal

3 - Reconnaissance

Normalization

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

In [39]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
features = train.iloc[:,:-1]
cols=features.columns
scaled_features= scaler.fit_transform(features)
train= pd.DataFrame(scaled_features,columns=cols)
```

In [40]:
```
features = test.iloc[:,:-1]
cols=features.columns
scaled_features= scaler.fit_transform(features)
test= pd.DataFrame(scaled_features,columns=cols)
```

In [41]:
```
train.head()
```

Out[41]:

	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	1.30
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	1.30
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	-1.28
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	1.30
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	-1.28

In [42]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2934752 entries, 0 to 2934751
Data columns (total 15 columns):
 #   Column             Dtype  
---  ------             -----  
 0   sport              float64
 1   dport              float64
 2   stddev             float64
 3   N_IN_Conn_P_SrcIP  float64
 4   min                float64
 5   state_number       float64
 6   mean               float64
 7   N_IN_Conn_P_DstIP  float64
 8   drate              float64
 9   srate              float64
 10  max                float64
 11  attack             float64
 12  saddr_enc          float64
 13  daddr_enc          float64
 14  proto_enc          float64
dtypes: float64(15)
memory usage: 335.9 MB
```

All the datatypes are float and all are in same scale.

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 under sampling 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.

In [45]:
```
import imblearn
from imblearn.over_sampling import RandomOverSampler
samp_strat= { 0 : 1541315, 1 : 1320148, 2 : 72919, 3 : 72919}
random_over= RandomOverSampler(sampling_strategy=samp_strat,random_state=1)
Xres,yres = random_over.fit_resample(train,y_train)
```

In [46]:
```
pd.Series(yres).value_counts()
```

Out[46]:
```
0    1541315
1    1320148
3      72919
2      72919
Name: category_enc, dtype: int64
```

In [47]:
```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
samp_strat= { 0 : 72919, 1 : 72919, 2 : 72919, 3 : 72919}
random_under= RandomUnderSampler(sampling_strategy=samp_strat,random_state=1)
Xres1,yres1 = random_under.fit_resample(Xres,yres)
```

In [48]:
```
pd.Series(yres1).value_counts()
```

Out[48]:
```
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 SVM Model.

In [44]:
```
from sklearn import model_selection
from sklearn.model_selection import StratifiedKFold,cross_val_score
from sklearn.svm import SVC
```

In [49]:
```
model = SVC(verbose=1,random_state=42)
model.fit(Xres1,yres1)
score = model.score(test,y_test)
score
```

```
[LibSVM]
```

Out[49]:
```
0.8990678637191951
```

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 StratifiedKFold and have used 10 splits. The score is calculated using the cross-validation score.

```
In [53]: model = SVC()
         skf = StratifiedKFold(n_splits=10,shuffle=True,random_state=42)
         scores = cross_val_score(model,Xres1,yres1,scoring='accuracy',cv=skf)

In [54]: scores

Out[54]: array([0.92594624, 0.93043747, 0.92995749, 0.92868897, 0.93006034,
                0.92892896, 0.92405801, 0.92659512, 0.92724655, 0.92830939])

In [55]: from numpy import mean
         mean(scores)

Out[55]: 0.9280228545291823
```

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 SVM we have used the parameter gamma as 0.01 and 0.1 and C as 1 and 10.

Best parameters C:10 and gamma 0.1

Best Score 0.9973

```
In [57]: params = {'gamma':[0.01,0.1],'C':(1,10)}
         from sklearn.model_selection import GridSearchCV
         model1 = SVC(random_state=42)
         gscv = GridSearchCV(model1,params,verbose=3,cv=3)
         gscv.fit(Xres1,yres1)
         print("best score:",gscv.best_score_)
         print("best params:",gscv.best_params_)

Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV] C=1, gamma=0.01 ...............................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] .................... C=1, gamma=0.01, score=0.985, total= 2.8min
[CV] C=1, gamma=0.01 ...............................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.8min remaining:    0.0s
[CV] .................... C=1, gamma=0.01, score=0.985, total= 2.9min
[CV] C=1, gamma=0.01 ...............................................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  5.7min remaining:    0.0s
[CV] .................... C=1, gamma=0.01, score=0.985, total= 2.8min
[CV] C=1, gamma=0.1 ................................................
[CV] ..................... C=1, gamma=0.1, score=0.993, total= 5.0min
[CV] C=1, gamma=0.1 ................................................
[CV] ..................... C=1, gamma=0.1, score=0.993, total= 5.2min
[CV] C=1, gamma=0.1 ................................................
[CV] ..................... C=1, gamma=0.1, score=0.993, total= 5.5min
[CV] C=10, gamma=0.01 ..............................................
[CV] ................... C=10, gamma=0.01, score=0.990, total= 1.4min
[CV] C=10, gamma=0.01 ..............................................
[CV] ................... C=10, gamma=0.01, score=0.990, total= 1.4min
[CV] C=10, gamma=0.01 ..............................................
[CV] ................... C=10, gamma=0.01, score=0.990, total= 1.3min
[CV] C=10, gamma=0.1 ...............................................
[CV] .................... C=10, gamma=0.1, score=0.997, total= 2.8min
[CV] C=10, gamma=0.1 ...............................................
[CV] .................... C=10, gamma=0.1, score=0.997, total= 2.8min
[CV] C=10, gamma=0.1 ...............................................
[CV] .................... C=10, gamma=0.1, score=0.997, total= 2.9min
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed: 36.9min finished
best score: 0.9973703695920392
best params: {'C': 10, 'gamma': 0.1}
```

Results

Model Score – 0.89906

```
In [49]: model = SVC(verbose=1,random_state=42)
         model.fit(Xres1,yres1)
         score = model.score(test,y_test)
         score

         [LibSVM]

Out[49]: 0.8990678637191951
```

Classification report

```
In [50]: from sklearn.metrics import classification_report,confusion_matrix
         prediction = model.predict(test)
         print(classification_report(y_test,prediction))

              precision    recall  f1-score   support

           0       0.89      0.93      0.91    385309
           1       1.00      0.86      0.92    330112
           2       1.00      0.99      1.00       107
           3       0.38      0.93      0.54     18163

    accuracy                           0.90    733691
   macro avg       0.82      0.93      0.84    733691
weighted avg       0.92      0.90      0.91    733691
```

Confusion Matrix

```
In [51]: print(confusion_matrix(y_test,prediction))

         [[359050    157      0  26102]
          [ 44781 283591      0   1740]
          [     0      0    106      1]
          [   339    933      0  16891]]
```

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

Logistic Regression: ~ 97%
SVM: ~ 99%
XGBoost: ~ 91%

Full Code

The full code is uploaded on GITHUB

GitHub Link - https://github.com/PseudoKush/Intrusion-Detection-System/blob/main/Intrusion%20Detection%20Using%20SVM.ipynb