NAME: AVIREDDY NVSRK ROHAN

REG.NO: 19BCE1180

DATASET USED: KDD CUP

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report
```

In [3]:
```
data= pd.read_csv("/content/drive/MyDrive/ML_LAB/corrected",header=None)
data.columns=['duration','protocol_type','service','flag','src_bytes','dst_bytes'
```

Data Preprocessing

In [4]:
```
cond=((data['class']=="normal.") | (data['class']=="neptune.") | (data['class']==
```

In [5]:
```
new_data = data[cond]
new_data["class"].value_counts()
```

Out[5]:
```
normal.           60593
neptune.          58001
snmpgetattack.     7741
Name: class, dtype: int64
```

Standardization

In [6]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
new_data['protocol_type'] = le.fit_transform(new_data['protocol_type'])
new_data['service'] = le.fit_transform(new_data['service'])
new_data['flag'] = le.fit_transform(new_data['flag'])
new_data['class'] = le.fit_transform(new_data['class'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  This is separate from the ipykernel package so we can avoid doing imports unt
il
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  """
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [7]:
```
target = new_data['class']
new_data=new_data.drop("class",axis=1)
```

In [8]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(new_data)
data_std=scaler.transform(new_data)
```

using smote to resample the imbalanced data

In [9]:
```
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
data_X, target_y = oversample.fit_resample(data_std,target)
```

In [ ]:
```
data_X.shape
```

Out[11]:
```
(181779, 41)
```

Train Test Split 80:20

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_X, target_y, test_size=0
```

ENSEMBLE=LogReg+MLP+KNN

Logistic Regression

In [13]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C=100.0, random_state=1, solver='lbfgs', multi_class='mul
lr.fit(X_train,y_train)

lr_pred = lr.predict(X_test)
print(classification_report(y_test,lr_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.80      0.89     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.93     36356
   macro avg       0.94      0.93      0.93     36356
weighted avg       0.94      0.93      0.93     36356
```

Multi Layer Perceptron

In [14]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(6, 2), random_
mlp.fit(X_train,y_train)

mlp_pred = mlp.predict(X_test)
print(classification_report(y_test,mlp_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.82      0.90     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.94     36356
   macro avg       0.95      0.94      0.94     36356
weighted avg       0.95      0.94      0.94     36356


/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perce
ptron.py:549: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

KNN

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(3)
kn.fit(X_train,y_train)

kn_pred = kn.predict(X_test)
print(classification_report(y_test,kn_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       0.95      0.89      0.92     12119
           2       0.90      0.96      0.93     12119

    accuracy                           0.95     36356
   macro avg       0.95      0.95      0.95     36356
weighted avg       0.95      0.95      0.95     36356
```

Hard Voting

In [16]:
```
mv_pred=[]
for i in range(len(lr_pred)):
  temp=[lr_pred[i],mlp_pred[i],kn_pred[i]]
  unique, counts = np.unique(np.array(temp), return_counts=True)
  mv_pred.append(unique[np.where(counts == np.max(counts))[0][0]])
```

In [17]:
```
print(classification_report(y_test,mv_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.81      0.90     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.94     36356
   macro avg       0.95      0.94      0.94     36356
weighted avg       0.95      0.94      0.94     36356
```

Stacking ensemble (knn-->log reg)

KNN (layer 1)

In [18]:
```
from sklearn.model_selection import KFold

kfold = KFold(10)

kn_pred1=[]
X=np.array(X_train)
y=np.array(y_train)
for train, test in kfold.split(X):
  X_train1, X_test1, y_train1, y_test1 = X[train], X[test], y[train], y[test]
  kn1 = KNeighborsClassifier(3)
  kn1.fit(X_train1,y_train1)
  kn_pred1.extend(kn1.predict(X_test1))
kn1.fit(X_train,y_train)
kn_test_pred=kn1.predict(X_test)
```

Logistic Regression (Level 2)

In [19]:
```
X1=np.insert(X_train,41,np.array(kn_pred1),axis=1)
```

In [20]:
```
lr1 = LogisticRegression(C=100.0, random_state=1, solver='lbfgs', multi_class='mu
lr1.fit(X1,y_train)
```

Out[20]:
```
LogisticRegression(C=100.0, max_iter=10000, multi_class='multinomial',
                   random_state=1)
```

Testing

In [21]:
```
X2 =np.insert(X_test,41,np.array(kn_test_pred),axis=1)
```

In [22]:
```
lr_pred2=lr1.predict(X2)
print(classification_report(y_test,lr_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       0.95      0.89      0.92     12119
           2       0.90      0.96      0.93     12119

    accuracy                           0.95     36356
   macro avg       0.95      0.95      0.95     36356
weighted avg       0.95      0.95      0.95     36356
```

Bagging

using kfold split (3 way) for classifiers

In [23]:
```
kfold1 = KFold(n_splits=3,shuffle=True,random_state=134)

X_train3=[]
y_train3=[]
X=np.array(X_train)
y=np.array(y_train)
for train, test in kfold1.split(X=X,y=y):
  X_train3.append(X[train])
  y_train3.append(y[train])
```

KNN

In [27]:
```
kn3 = KNeighborsClassifier(3)
kn3.fit(X_train3[0],y_train3[0])

kn_pred3 = kn3.predict(X_test)
print(classification_report(y_test,kn_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       0.96      0.89      0.93     12119
           2       0.90      0.97      0.93     12119

    accuracy                           0.95     36356
   macro avg       0.95      0.95      0.95     36356
weighted avg       0.95      0.95      0.95     36356
```

Logistic Regression

In [28]:
```
lr3 = LogisticRegression(C=100.0, random_state=1, solver='lbfgs', multi_class='mu
lr3.fit(X_train3[1],y_train3[1])

lr_pred3 = lr3.predict(X_test)
print(classification_report(y_test,lr_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.80      0.89     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.93     36356
   macro avg       0.94      0.93      0.93     36356
weighted avg       0.94      0.93      0.93     36356
```

MLP

In [29]:
```
mlp3 = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random
mlp3.fit(X_train3[2],y_train3[2])

mlp_pred3 = mlp2.predict(X_test)
print(classification_report(y_test,mlp_pred3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.81      0.89     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.94     36356
   macro avg       0.95      0.94      0.94     36356
weighted avg       0.95      0.94      0.94     36356


/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perce
ptron.py:549: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

Majority Voting

In [31]:
```
mv_pred=[]
for i in range(len(lr_pred3)):
  temp=[lr_pred3[i],mlp_pred3[i],kn_pred3[i]]
  unique, counts = np.unique(np.array(temp), return_counts=True)
  mv_pred.append(unique[np.where(counts == np.max(counts))[0][0]])
```

In [32]:
```
print(classification_report(y_test,mv_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12118
           1       1.00      0.81      0.89     12119
           2       0.84      1.00      0.91     12119

    accuracy                           0.94     36356
   macro avg       0.95      0.94      0.94     36356
weighted avg       0.95      0.94      0.94     36356
```