CSC 215-01 Artificial Intelligence (Fall 2018)

Mini-Project 3: Network Intrusion Detection

Due at 4 pm, Monday, October 22, 2018

(1) Problem Statement

- To detect the network intrusions and predict where it is bad or good connection using different sklearn Models, Neural Network and CNN.

(2) Methodology

-Converted a CSV file into a Dataframe

```
df = pd.read_csv('network_intrusion_data.csv',encoding = 'ISO-8859-1',header=None,names=column_names)
```

-Removed rows with any NULL values

```
df_clean = df.dropna() #drop any null value row
```

-Removed all duplicate values

```
df_unique = df_clean.drop_duplicates(keep='first', inplace=False) #removing dulicates
```

-Performed Binary Classification (i.e 0-Good connection and 1-Bad connection)

```
def binary_label_encoding(label):
    if label=='normal.':
        return 0
    else:
        return 1
```

```
df_unique['binary_label']=df_unique.outcome.apply(binary_label_encoding)
```

-Performed one hot encoding for the columns : protocol_type, service, flag and su_ attempted.

```
encode_text_dummy(df_unique,'protocol_type')
```

```
encode_text_dummy(df_unique,'service')
```

.

```
encode_text_dummy(df_unique,'flag')
```

```
encode_text_dummy(df_unique,'su_attempted')
```

-Performed min max normalization for the numeric features

```
numerical_features = ['duration','src_bytes','dst_bytes','wrong_fragment','urgent','hot','num_failed_logins',
'num_compromised','num_root','num_file_creations','num_shells','num_access_files','count',
'srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate',
'same_srv_rate',
'diff_srv_rate',
'srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate',
'dst_host_diff_srv_rate',
'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate',
'dst_host_serror_rate',
'dst_host_srv_serror_rate',
'dst_host_rerror_rate',
'dst_host_srv_rerror_rate']
```

```
for name in numerical_features:
    min_max_normalization(df_unique,name)
```

-Deleted is_host_login column and num_outbound_cmds column as the values is these columns have one unique value and by removing these columns will not change the results of the output.

-Splitted the data into Train and Test.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y_binary,test_size=0.20,random_state=1)
```

-Trained all the models: SVM, KNN, Logistic Regression, Gaussian Naïve Bayes, Neural Networks and CNN(Covolutional Neural Networks).

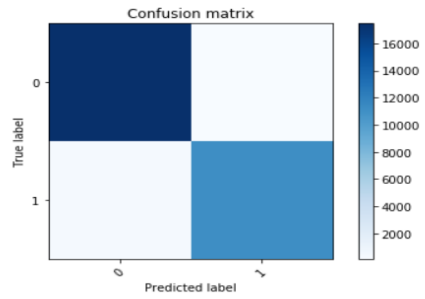
-Predicted all models with their F-1scores, Confusion matrix and ROC curve.

(3) Experimental Results and Analysis

SVM

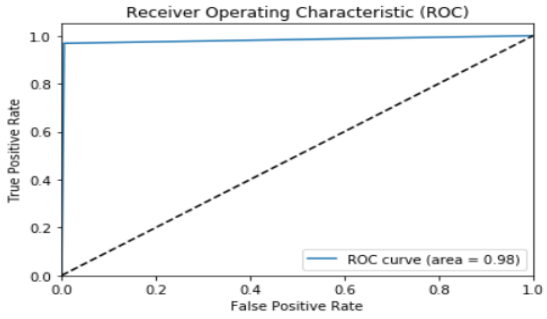
```
y_pred_svm = clf_svc.predict(X_test)

print("test : ", y_test[:10])
print("pred : ", y_pred_svm[:10])

print()

cm_svm = confusion_matrix(y_test, y_pred_svm)
print(cm_svm)

print()

print("Precision Score:: ",metrics.precision_score(y_test,y_pred_svm))
print("Recall Score::    ",metrics.recall_score(y_test,y_pred_svm))
print("F1 Score::        ",metrics.f1_score(y_test,y_pred_svm))

print()

print('Plotting confusion matrix')
plt.figure()
plot_confusion_matrix(cm_svm, clf_svc.classes_)
plt.show()

print()

print(classification_report(y_test, y_pred_svm))
```

```
test :  [0 1 0 1 0 1 0 0 0 1]
pred :  [0 1 0 1 0 1 0 0 0 1]

[[17463    96]
 [  367 11192]]

Precision Score::  0.9914953933380581
Recall Score::     0.9682498486028203
F1 Score::         0.9797347572985513
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     17559
           1       0.99      0.97      0.98     11559
```

```
plot_roc(y_pred_svm,y_test)
```

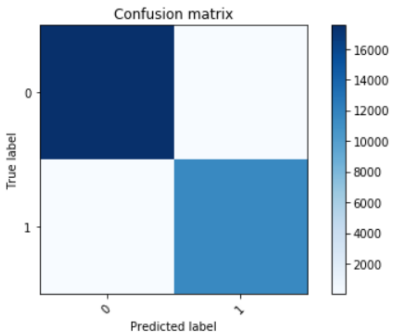


KNN

```
y_pred_knn = clf_knn.predict(X_test)

print("test : ", y_test[:10])
print("pred : ", y_pred_knn[:10])

print()

cm_knn = confusion_matrix(y_test, y_pred_knn)
print(cm_knn)

print()

print("Precision Score:: ",metrics.precision_score(y_test,y_pred_knn))
print("Recall Score   :: ",metrics.recall_score(y_test,y_pred_knn))
print("F1 Score       :: ",metrics.f1_score(y_test,y_pred_knn))

print()

print('Plotting confusion matrix')
plt.figure()
plot_confusion_matrix(cm_knn, clf_knn.classes_)
plt.show()

print()

print(classification_report(y_test, y_pred_knn))
```

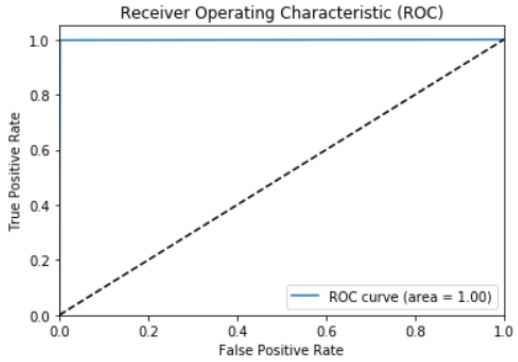
```
test :  [0 1 0 1 0 1 0 0 0 1]
pred :  [0 1 0 1 0 1 0 0 0 1]

[[17536    23]
 [   32 11527]]

Precision Score::  0.998008658008658
Recall Score   ::  0.9972315944285838
F1 Score       ::  0.9976199749015536
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
plot_roc(y_pred_knn,y_test)
```

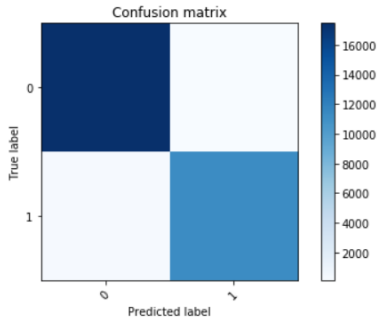


Logistic Regression

```
y_pred_lr = clf_lr.predict(X_test)

print("test : ", y_test[:10])
print("pred : ", y_pred_lr[:10])

print()

cm_lr = confusion_matrix(y_test, y_pred_lr)
print(cm_lr)

print()

print("Precision Score:: ",metrics.precision_score(y_test,y_pred_lr))
print("Recall Score   :: ",metrics.recall_score(y_test,y_pred_lr))
print("F1 Score       :: ",metrics.f1_score(y_test,y_pred_lr))

print()

print('Plotting confusion matrix')
plt.figure()
plot_confusion_matrix(cm_lr, clf_lr.classes_)
plt.show()

print()

print(classification_report(y_test, y_pred_lr))
```

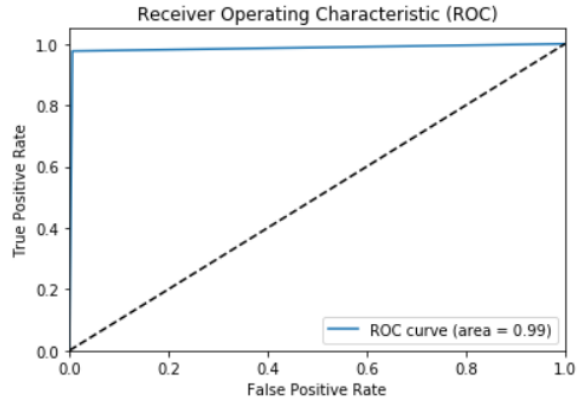
```
test :  [0 1 0 1 0 1 0 0 0 1]
pred :  [0 1 0 1 0 1 0 0 0 1]

[[17451   108]
 [  272 11287]]

Precision Score::  0.9905221588415972
Recall Score   ::  0.9764685526429622
F1 Score       ::  0.9834451511719091
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     17559
           1       0.99      0.98      0.98     11559
```

```
plot_roc(y_pred_lr,y_test)
```

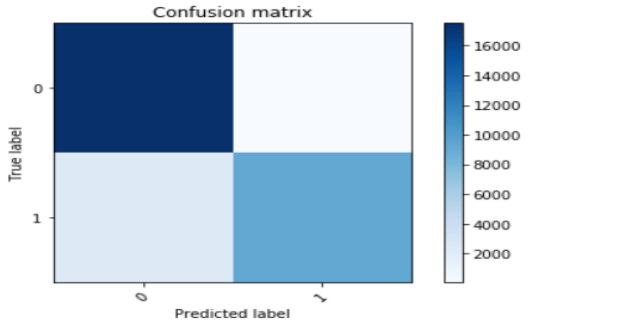


Gaussian Naïve Bayes

```
y_pred_gnb = clf_gnb.predict(X_test)

print("test : ", y_test[:10])
print("pred : ", y_pred_gnb[:10])

print()

cm_gnb = confusion_matrix(y_test, y_pred_gnb)
print(cm_gnb)

print()

print("Precision Score:: ",metrics.precision_score(y_test,y_pred_gnb))
print("Recall Score   :: ",metrics.recall_score(y_test,y_pred_gnb))
print("F1 Score       :: ",metrics.f1_score(y_test,y_pred_gnb))

print()

print('Plotting confusion matrix')
plt.figure()
plot_confusion_matrix(cm_gnb, clf_gnb.classes_)
plt.show()

print()

print(classification_report(y_test, y_pred_gnb))
```

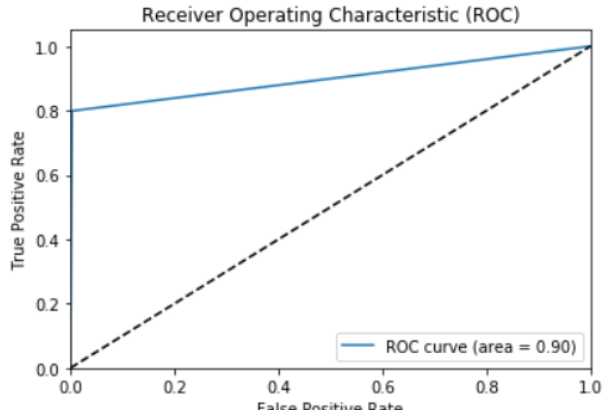
```
test :  [0 1 0 1 0 1 0 0 0 1]
pred :  [0 0 0 1 0 1 0 0 0 1]

[[17508    51]
 [ 2325  9234]]

Precision Score::  0.9945072697899838
Recall Score   ::  0.7988580327017908
F1 Score       ::  0.8860103626943006
```

```
Plotting confusion matrix
```



	precision	recall	f1-score	support
0	0.88	1.00	0.94	17559
1	0.99	0.80	0.89	11559

```
plot_roc(y_pred_gnb,y_test)
```



Classification Models Analysis			
Model	**F1 score**	**Precision score**	**Recall Score**
SVM	0.97	0.99	0.96
KNN	0.99	0.99	0.99
Logistic Regression	0.98	0.99	0.97
Gaussian Naïve Bayes	0.88	0.99	0.79
Best Neural Network model in Tensorflow	0.998	0.998	0.997
Best Convolutional Neural Network model in Tensorflow	0.9959	0.996	0.995

	Neural Network Analysis			
Activation Layer	**Optimizer**	**F1 score**	**Precision score**	**Recall Score**
relu	adam	0.997	0.998	0.997
sigmoid	adam	0.997	0.998	0.996
tanh	adam	0.998	0.998	0.997
relu	sgd	0.997	0.998	s0.996
sigmoid	sgd	0.982	0.989	0.975
tanh	sgd	0.996	0.997	0.996
relu	rmsprop	0.996	0.998	0.995
sigmoid	rmsprop	0.995	0.995	0.995
tanh	rmsprop	0.997	0.998	0.996

CNN parameter tuning analysis							
Model	**Kernel No.**	**Kernel Size**	**Strides**	**Activation**	**F-1 Score**	**Precision Score**	**Recall Score**
Model	32	(1,3)	(1,1)	relu	0.9958	0.996	0.995
	64	(1,3)	(1,1)	relu			
Model_1	32	(1,5)	(1,1)	sigmoid	0.97	0.99	0.95
	64	(1,5)	(1,1)	sigmoid			
Model_2	64	(1,5)	(1,2)	relu	0.993	0.993	0.992
	128	(1,5)	(1,2)	relu			
Model_3	64	(1,3)	(1,2)	relu	0.9959	0.996	0.995
	128	(1,3)	(1,2)	relu			
Model_4	64	(1,3)	(1,2)	tanh	0.994	0.995	0.993
	100	(1,3)	(1,2)	tanh			
Model_5	64	(1,2)	(1,2)	relu	0.994	0.993	0.994
	128	(1,2)	(1,2)	relu			
	256	(1,2)	(1,2)	relu			

→For Neural Network we have tried different parameters to achieve the BEST F1 score:

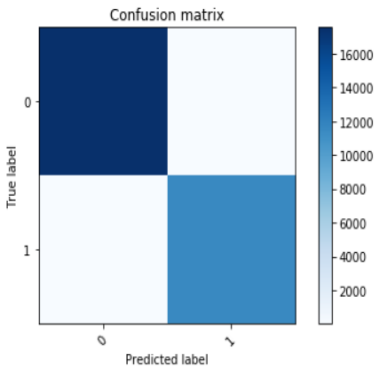
i) 2 hidden layers with Activation=”relu” and Optimizer=”adam”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification = Sequential()
    model_classification.add(Dense(50,input_dim=X_tns_train.shape[1], activation='relu')) # Hidden 1     #  why input_dim=x.shape
    model_classification.add(Dense(50,activation='relu')) # Hidden 2
    model_classification.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification.compile(loss='categorical_crossentropy', optimizer='adam')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],ve
```

```
model_classification.load_weights('class_weights/best_weights.hdf5')

pred = model_classification.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

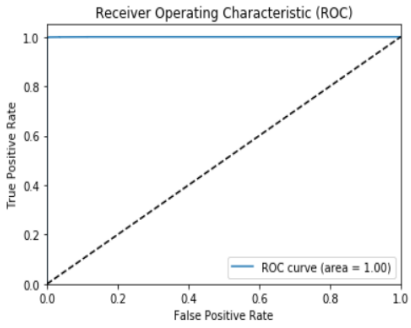
```
Precision score : 0.9984410185345575
Recall score    : 0.9973181071026905
F1 score        : 0.9978792469162518
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification.load_weights('class_weights/best_weights.hdf5')

pred = model_classification.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



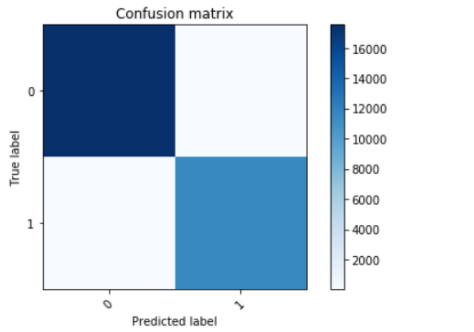
ii) 2 hidden layers with Activation=”sigmoid” and Optimizer=”adam”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_1.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_1 = Sequential()
    model_classification_1.add(Dense(50,input_dim=X_tns_train.shape[1], activation='sigmoid')) # Hidden 1     #  why input_dim=x
    model_classification_1.add(Dense(50,activation='sigmoid')) # Hidden 2
    model_classification_1.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_1.compile(loss='categorical_crossentropy', optimizer='adam')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_1.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer]
```

```
model_classification_1.load_weights('class_weights/best_weights_1.hdf5')

pred = model_classification_1.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

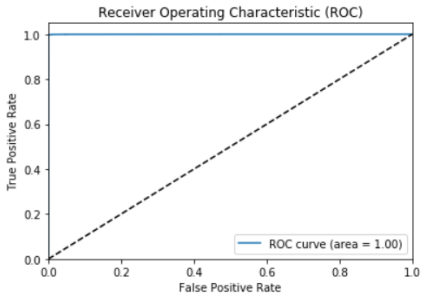
```
Precision score : 0.998267348176384
Recall score    : 0.9968855437321568
F1 score        : 0.9975759674487057
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_1.load_weights('class_weights/best_weights_1.hdf5')

pred = model_classification_1.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



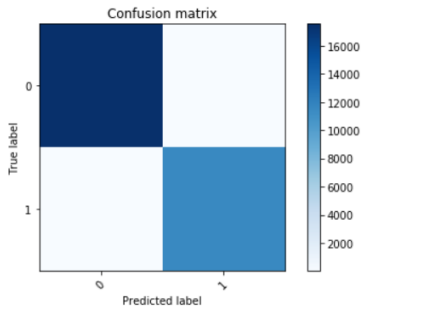
iii) 2 hidden layers with Activation=”tanh” and Optimizer=”tanh”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_2.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_2 = Sequential()
    model_classification_2.add(Dense(50,input_dim=X_tns_train.shape[1], activation='tanh')) # Hidden 1     #  why input_dim=x.sha
    model_classification_2.add(Dense(50,activation='tanh')) # Hidden 2
    model_classification_2.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_2.compile(loss='categorical_crossentropy', optimizer='adam')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_2.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

```
model_classification_2.load_weights('class_weights/best_weights_2.hdf5')

pred = model_classification_2.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

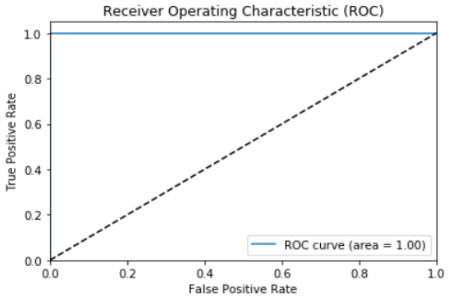
```
Precision score : 0.998700511132288
Recall score    : 0.9973181071026905
F1 score        : 0.9980088304042939
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_2.load_weights('class_weights/best_weights_2.hdf5')

pred = model_classification_2.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



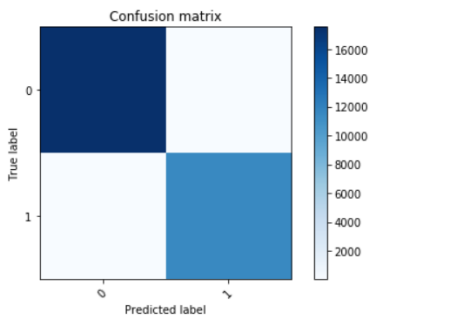
iv) 2 hidden layers with Activation=”relu” and Optimizer=”sgd”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_3.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_3 = Sequential()
    model_classification_3.add(Dense(50,input_dim=X_tns_train.shape[1], activation='relu')) # Hidden 1     #  why input_dim=x.sha
    model_classification_3.add(Dense(50,activation='relu')) # Hidden 2
    model_classification_3.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_3.compile(loss='categorical_crossentropy', optimizer='sgd')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_3.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

```
model_classification_3.load_weights('class_weights/best_weights_3.hdf5')

pred = model_classification_3.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

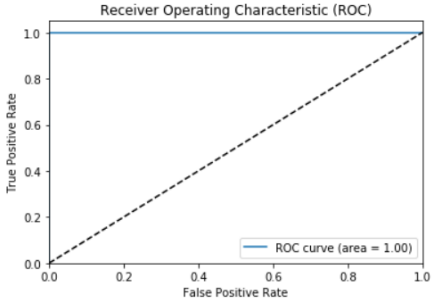
```
Precision score : 0.998093917865188
Recall score    : 0.9966260057098365
F1 score        : 0.9973594216700576
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_3.load_weights('class_weights/best_weights_3.hdf5')

pred = model_classification_3.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



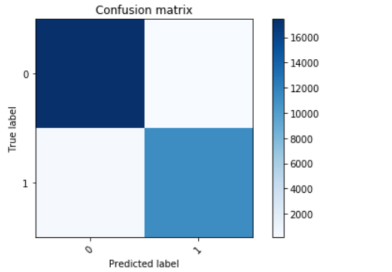
v) 2 hidden layers with Activation=”sigmoid” and Optimizer=”sgd”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_4.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_4 = Sequential()
    model_classification_4.add(Dense(50,input_dim=X_tns_train.shape[1], activation='sigmoid')) # Hidden 1     #  why input_dim=x
    model_classification_4.add(Dense(50,activation='sigmoid')) # Hidden 2
    model_classification_4.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_4.compile(loss='categorical_crossentropy', optimizer='sgd')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_4.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer]
```

```
model_classification_4.load_weights('class_weights/best_weights_4.hdf5')

pred = model_classification_4.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

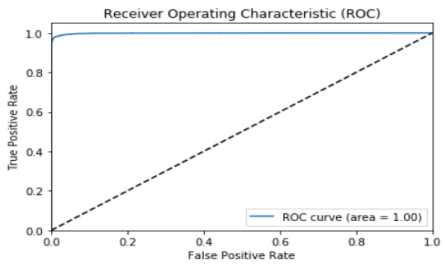
```
Precision score : 0.989640035118525
Recall score    : 0.9751708625313609
F1 score        : 0.9823521722079394
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     17559
           1       0.99      0.98      0.98     11559
```

```
model_classification_4.load_weights('class_weights/best_weights_4.hdf5')

pred = model_classification_4.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



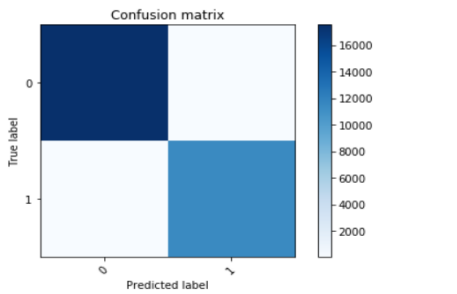
vi) 2 hidden layers with Activation=”tanh” and Optimizer=”sgd”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_5.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_5 = Sequential()
    model_classification_5.add(Dense(50,input_dim=X_tns_train.shape[1], activation='tanh')) # Hidden 1     #  why input_dim=x.sha
    model_classification_5.add(Dense(50,activation='tanh')) # Hidden 2
    model_classification_5.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_5.compile(loss='categorical_crossentropy', optimizer='sgd')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_5.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

```
model_classification_5.load_weights('class_weights/best_weights_5.hdf5')

pred = model_classification_5.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

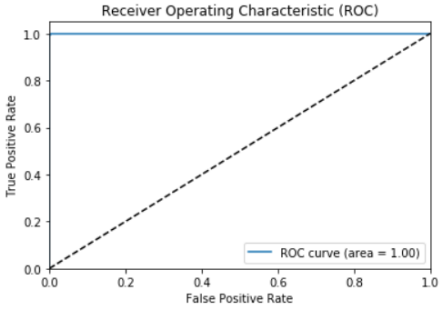
```
Precision score : 0.9976605146867689
Recall score    : 0.996106929665196
F1 score        : 0.9968831168831168
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_5.load_weights('class_weights/best_weights_5.hdf5')

pred = model_classification_5.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



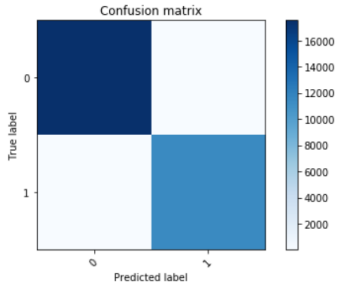
vii) 2 hidden layers with Activation=”relu” and Optimizer=”rmsprop”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_6.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_6 = Sequential()
    model_classification_6.add(Dense(50,input_dim=X_tns_train.shape[1], activation='relu')) # Hidden 1     #  why input_dim=x.sha
    model_classification_6.add(Dense(50,activation='relu')) # Hidden 2
    model_classification_6.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_6.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_6.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

```
model_classification_6.load_weights('class_weights/best_weights_6.hdf5')

pred = model_classification_6.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

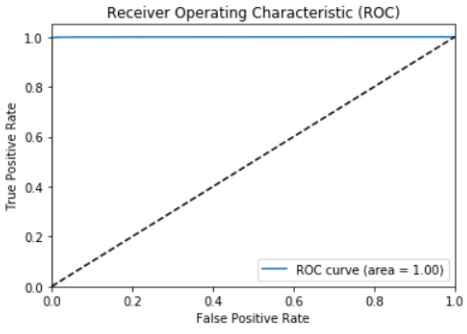
```
Precision score : 0.9980919340849956
Recall score    : 0.9955878536205555
F1 score        : 0.9968383212785309
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_6.load_weights('class_weights/best_weights_6.hdf5')

pred = model_classification_6.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



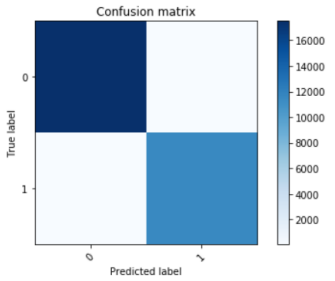
viii) 2 hidden layers with Activation=”sigmoid” and Optimizer=”rmsprop”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_7.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_7 = Sequential()
    model_classification_7.add(Dense(50,input_dim=X_tns_train.shape[1], activation='sigmoid')) # Hidden 1     #  why input_dim=x.
    model_classification_7.add(Dense(50,activation='sigmoid')) # Hidden 2
    model_classification_7.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_7.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_7.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

```
model_classification_7.load_weights('class_weights/best_weights_7.hdf5')

pred = model_classification_7.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

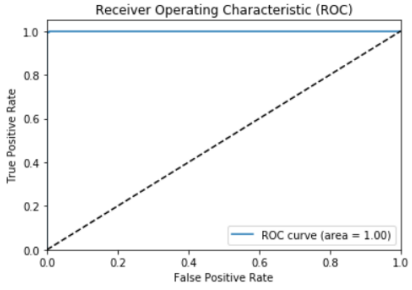
```
Precision score : 0.9954156214860306
Recall score    : 0.9955878536205555
F1 score        : 0.9955017301038063
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_7.load_weights('class_weights/best_weights_7.hdf5')

pred = model_classification_7.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```

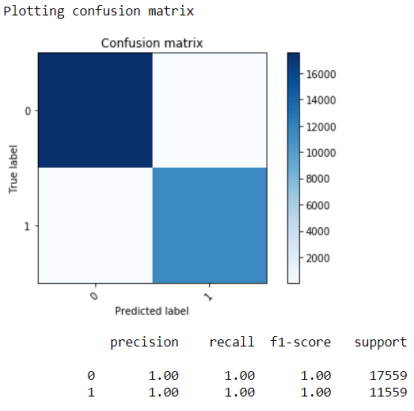


ix) 2 hidden layers with Activation=”tanh” and Optimizer=”rmsprop”

```
checkpointer = ModelCheckpoint(filepath="class_weights/best_weights_8.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(5):
    print(i)
    model_classification_8 = Sequential()
    model_classification_8.add(Dense(50,input_dim=X_tns_train.shape[1], activation='tanh')) # Hidden 1     #  why input_dim=x.sha
    model_classification_8.add(Dense(50,activation='tanh')) # Hidden 2
    model_classification_8.add(Dense(y_tns_train.shape[1],activation='softmax')) # Output
    model_classification_8.compile(loss='categorical_crossentropy', optimizer='rmsprop')

    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto')
    model_classification_8.fit(X_tns_train, y_tns_train,validation_data=(X_tns_test,y_tns_test),callbacks=[monitor,checkpointer],
```

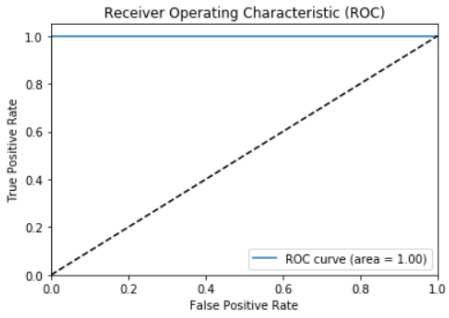
```
model_classification_8.load_weights('class_weights/best_weights_8.hdf5')

pred = model_classification_8.predict(X_tns_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)
y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score : {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score    : {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score        : {}".format(f1_score))

print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(classification_report(y_true, pred))
```

```
Precision score : 0.9986127969481533
Recall score    : 0.996452980361623
F1 score        : 0.9975317195687003
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559
```

```
model_classification_8.load_weights('class_weights/best_weights_8.hdf5')

pred = model_classification_8.predict(X_tns_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



→For CNN we have tried different parameters to achieve the BEST F1 score:

The best f-1 score is generated from Model_3.

```
cnn_model_3 = Sequential()
cnn_model_3.add(Conv2D(64, kernel_size=(1, 3), strides=(1, 2),activation='relu',input_shape=input_shape,padding='same'))
cnn_model_3.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
cnn_model_3.add(Conv2D(128, kernel_size=(1, 3), strides=(1, 2),activation='relu',padding='same'))
cnn_model_3.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
cnn_model_3.add(Flatten())
cnn_model_3.add(Dense(500, activation='relu'))
cnn_model_3.add(Dense(y_tns_train.shape[1], activation='softmax'))
cnn_model_3.summary()
```

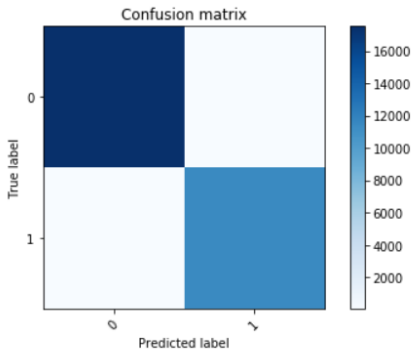
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 1, 59, 64)         256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 1, 30, 64)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 1, 15, 128)        24704
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 8, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_142 (Dense)            (None, 500)               512500
_________________________________________________________________
dense_143 (Dense)            (None, 2)                 1002
=================================================================
Total params: 538,462
Trainable params: 538,462
Non-trainable params: 0
```

```
checkpointer = ModelCheckpoint(filepath="cnn_weights/best_weights_3.hdf5", verbose=0, save_best_only=True) # save best model

for i in range(3):
    print(i)
    cnn_model_3.compile(loss='categorical_crossentropy', optimizer='adam')
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=2, verbose=1, mode='auto')
    cnn_model_3.fit(X_tns_re_train[:5000], y_tns_train[:5000],batch_size=128,validation_data=(X_tns_re_test[:1000],y_tns_test[:10
```

```
cnn_model_3.load_weights('cnn_weights/best_weights_3.hdf5')

pred = cnn_model_3.predict(X_tns_re_test)

pred = np.argmax(pred,axis=1) # raw probabilities to chosen class (highest probability)

y_true= np.argmax(y_tns_test,axis=1)

pr_score = metrics.precision_score(y_true, pred)
print("Precision score: {}".format(pr_score))

re_score = metrics.recall_score(y_true, pred)
print("Recall score: {}".format(re_score))

f1_score = metrics.f1_score(y_true, pred)
print("F1 score: {}".format(f1_score))


print()

cm = confusion_matrix(y_true, pred)
print(cm)

print()

print('Plotting confusion matrix')

plt.figure()
plot_confusion_matrix(cm, clf_knn.classes_)
plt.show()

print(metrics.classification_report(y_true, pred))
```

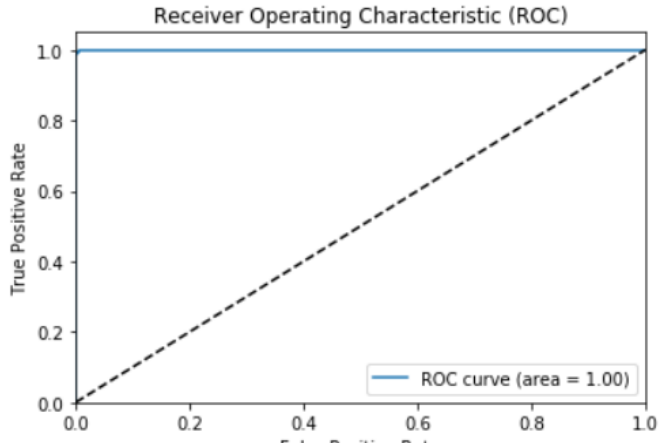
```
Precision score: 0.9963636363636363
Recall score: 0.9955878536205555
F1 score: 0.9959755939244451
```

```
Plotting confusion matrix
```




```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17559
           1       1.00      1.00      1.00     11559

   micro avg       1.00      1.00      1.00     29118
   macro avg       1.00      1.00      1.00     29118
weighted avg       1.00      1.00      1.00     29118
```

```
cnn_model_3.load_weights('cnn_weights/best_weights_3.hdf5')

pred = cnn_model_3.predict(X_tns_re_test)

pred = pred[:,1] # Only positive class (1)
plot_roc(pred,y_true)
```



4. TASK DIVISION AND PROJECT REFLECTION

Name: Jinaliben Shah **Student Id:** 219209290

Tasks performed:
Removed rows with any null values.
Removed duplicate rows
Encoded Categorical features
Implemented 2 models (Nearest Neighbor, Gaussian Naïve Bayes).
Confusion Matrix for each model.
Neural Network and the Parameter Tuning for Classification Models.

Name: Mardavkumar Gandhi **Student Id:** 219225917

Tasks performed:
Normalized Numeric features using min max normalization.
Split the data into train and test data.
Implemented 2 models (Support Vector Machine, Logistic Regression)
Prediction for the Test data and compared actual and predicted result.
ROC Curve for each model.
Convolutional Neural Network and Parameter Tuning for CNN (Kernel size, Kernel no. and Strides)

LEARNING:

What we have learnt from this project:

- How to do feature normalization (min-max scaling).
- Numpy , pandas and various operations on numpy array and dataframe
- **Applying the models and generating their scores and comparing their performance.**
- **How to apply Gaussian Naïve Bayes Model.**
- How to implement **neural network using tensorflow and keras.**
- How to use **Early stopping** and **Save and Use saved best weights of Neural Networks.**
- Parameter Tuning of Neural Networks like **optimizer** , **no. of hidden layers** , **no. of neurons in each layer** and different **activation functions.**
- How to implement Convolutional Neural Networks.
- Parameter Tuning for CNN(Kernel size, Kernel no. and Strides)

5. Extra features

- ➔ Performed multi-class classification with all 23 types of intrusions.
- ➔ Below is the analysis for multi-class classification for all sklearn models and Neural Network.
- ➔ Detailed insight of model , how they are performing for each class can be seen in notebook where we have printed classification Report.

Multiclass Classification Models Analysis			
Model	**F1 score**	**Precision score**	**Recall Score**
SVM	0.9836	0.9868	0.9873
KNN	0.9975	0.9973	0.9977
Logistic Regression	0.9867	0.9887	0.9893
Gaussian Naïve Bayes	0.5498	0.9710	0.4597
Best Neural Network model in Tensorflow	0.9983	0.9982	0.9984

	Neural Network Analysis			
Activation Layer	**Optimizer**	**F1 score**	**Precision score**	**Recall Score**
relu	adam	0.9983	0.9982	0.9984
sigmoid	adam	0.9977	0.9979	0.9976
tanh	adam	0.9978	0.9977	0.9979

Best Neural Network Classification Report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	221
1	1.00	1.00	1.00	3
2	1.00	0.67	0.80	3
3	0.93	0.93	0.93	14
5	0.99	0.99	0.99	124
6	1.00	1.00	1.00	6
7	0.00	0.00	0.00	1
8	0.00	0.00	0.00	4
9	1.00	1.00	1.00	10348
10	1.00	1.00	1.00	40
11	1.00	1.00	1.00	17559
12	1.00	1.00	1.00	1
14	1.00	1.00	1.00	36
15	1.00	0.99	0.99	78
16	0.00	0.00	0.00	4
17	0.99	0.97	0.98	186
18	0.99	0.99	0.99	118
20	1.00	1.00	1.00	191
21	0.95	0.93	0.94	178
22	0.67	0.67	0.67	3
micro avg	1.00	1.00	1.00	29118
macro avg	0.83	0.81	0.81	29118
weighted avg	1.00	1.00	1.00	29118