Importing relevant libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error,accura
import math
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.metrics import precision_score, recall_score, confusion_matrix, class
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

Setting headers (as given in pdf)

```
headers ="""duration,
protocol_type,
service,
flag,
src_bytes,
dst_bytes,
land,
wrong_fragment,
urgent,
hot,
num_failed_logins,
logged_in,
num_compromised,
root_shell,
su_attempted,
num_root,
num_file_creations,
num_shells,
num_access_files,
num_outbound_cmds,
is_host_login,
is_guest_login,
count,
srv_count,
serror_rate
```

```
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"""

columns =headers.split(",")
for i in range(len(columns)):
    columns[i]=columns[i].strip("\n")
```

Loading training data

```
columns.append('target')
print("number of features present in dataset",len(columns))
training_df = pd.read_csv("/content/drive/My Drive/full.csv",names=columns)
training_df
```

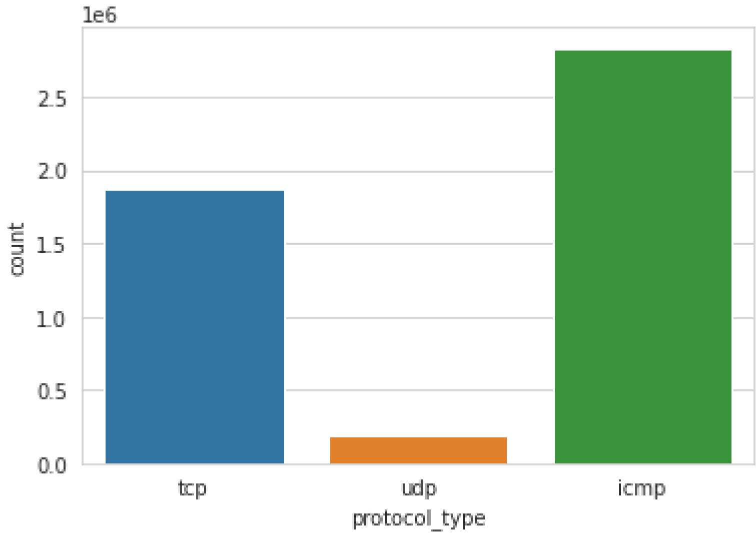
```
number of features present in dataset 42
```

duration protocol_type service flag src_bytes dst_bytes land wr

Plotting data count of protocols in training data

```
sns.set_style('whitegrid')
sns.countplot(x='protocol_type', data=training_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4127e4d810>
```



Loading testing data

```
testing_df = pd.read_csv("/content/drive/My Drive/test.csv",names=columns[:-1])
testing_df
```

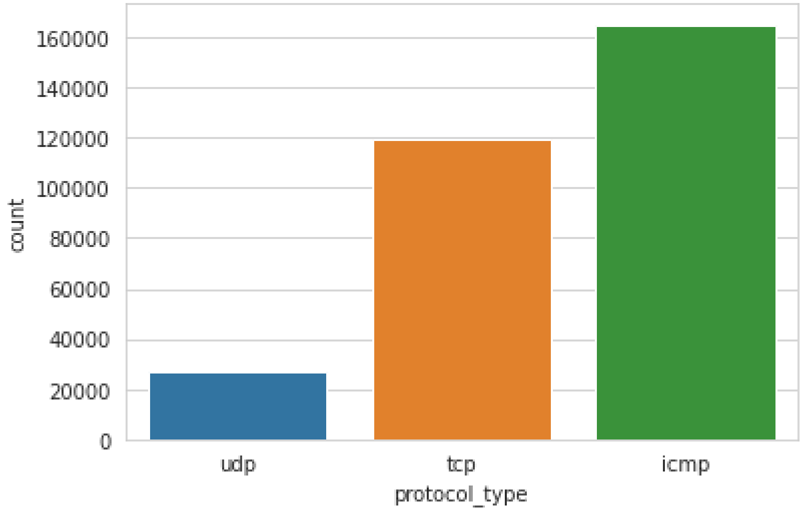
```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2718:
  interactivity=interactivity, compiler=compiler, result=result)
```

duration protocol_type service flag src_bytes dst_bytes land wr

Plotting data counts of each protocol in testing data

```
sns.set_style('whitegrid')
sns.countplot(x='protocol_type', data=testing_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f41112b09d0>
```



Dropping index

```
training_df.reset_index(drop=True,inplace=True)
testing_df.reset_index(drop=True,inplace=True)
```

Printing the unique values of our target column

```
unique_target = training_df["target"].unique()
print("Unique values in target column  (",len(unique_target)," classes ) :\n")
for val in unique_target:
  print(val)
```

```
Unique values in target column  ( 23  classes ) :

normal.
buffer_overflow.
loadmodule.
perl.
neptune.
smurf.
guess_passwd.
pod.
teardrop.
portsweep.
ipsweep.
```

```
land.
ftp_write.
back.
imap.
satan.
phf.
nmap.
multihop.
warezmaster.
warezclient.
spy.
rootkit.
```

Describing training data in details

```
training_df.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	
count	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.8984
mean	4.834243e+01	1.834621e+03	1.093623e+03	5.716116e-06	6.487792e-04	7.961
std	7.233298e+02	9.414311e+05	6.450123e+05	2.390833e-03	4.285434e-02	7.215
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
25%	0.000000e+00	4.500000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
50%	0.000000e+00	5.200000e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
75%	0.000000e+00	1.032000e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.0000
max	5.832900e+04	1.379964e+09	1.309937e+09	1.000000e+00	3.000000e+00	1.4000

Checking for missing values in the dataset

```
training_df.isnull().sum()
```

```
duration                0
protocol_type           0
service                 0
flag                    0
src_bytes               0
dst_bytes               0
land                    0
wrong_fragment          0
urgent                  0
hot                     0
num_failed_logins       0
logged_in               0
num_compromised         0
root_shell              0
su_attempted            0
```

```
num_root                       0
num_file_creations             0
num_shells                     0
num_access_files               0
num_outbound_cmds              0
is_host_login                  0
is_guest_login                 0
count                          0
srv_count                      0
serror_rate                    0
srv_serror_rate                0
rerror_rate                    0
srv_rerror_rate                0
same_srv_rate                  0
diff_srv_rate                  0
srv_diff_host_rate             0
dst_host_count                 0
dst_host_srv_count             0
dst_host_same_srv_rate         0
dst_host_diff_srv_rate         0
dst_host_same_src_port_rate    0
dst_host_srv_diff_host_rate    0
dst_host_serror_rate           0
dst_host_srv_serror_rate       0
dst_host_rerror_rate           0
dst_host_srv_rerror_rate       0
target                         0
dtype: int64
```

Describing test data in details

```
testing_df.describe()
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	
count	311029.000000	3.110290e+05	3.110290e+05	311029.000000	311029.000000	311(
mean	17.902736	1.731702e+03	7.479937e+02	0.000029	0.000763	
std	407.644400	1.276567e+05	1.612018e+04	0.005382	0.040369	
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	
25%	0.000000	1.050000e+02	0.000000e+00	0.000000	0.000000	
50%	0.000000	5.200000e+02	0.000000e+00	0.000000	0.000000	
75%	0.000000	1.032000e+03	0.000000e+00	0.000000	0.000000	
max	57715.000000	6.282565e+07	5.203179e+06	1.000000	3.000000	

Getting numeric data from training dataset

```
num_cols = training_df._get_numeric_data().columns
print("numeric data containing columns are (", len(num_cols)," columns ) :\n")
```

```
print(list(num_cols))
```

```
numeric data containing columns are ( 38  columns ) :

['duration', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'h
```

Getting categorical data from training data

```
category_cols=list(set(training_df.columns)-set(num_cols))
print("categorical data containing columns are (", len(category_cols)," columns )
print(category_cols)
```

```
categorical data containing columns are ( 4  columns ) :

['protocol_type', 'target', 'service', 'flag']
```

Dropping columns from training and testing data where value is NaN

```
training_df.dropna('columns',inplace=True)
testing_df.dropna('columns',inplace=True)
```

Getting columns with more than one value

```
cols=[col for col in training_df if training_df[col].nunique() > 1]
cols
```

```
['duration',
 'protocol_type',
 'service',
 'flag',
 'src_bytes',
 'dst_bytes',
 'land',
 'wrong_fragment',
 'urgent',
 'hot',
 'num_failed_logins',
 'logged_in',
 'num_compromised',
 'root_shell',
 'su_attempted',
 'num_root',
 'num_file_creations',
 'num_shells',
 'num_access_files',
 'is_host_login',
 'is_guest_login',
 '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',
     'target']
```

Dropping columns with only single value

```
list(set(training_df.columns)-set(cols))
training_df.drop('num_outbound_cmds', axis = 1, inplace = True)
testing_df.drop('num_outbound_cmds', axis = 1, inplace = True)
```

Displaying reduced number of features

```
len(training_df.columns)
```

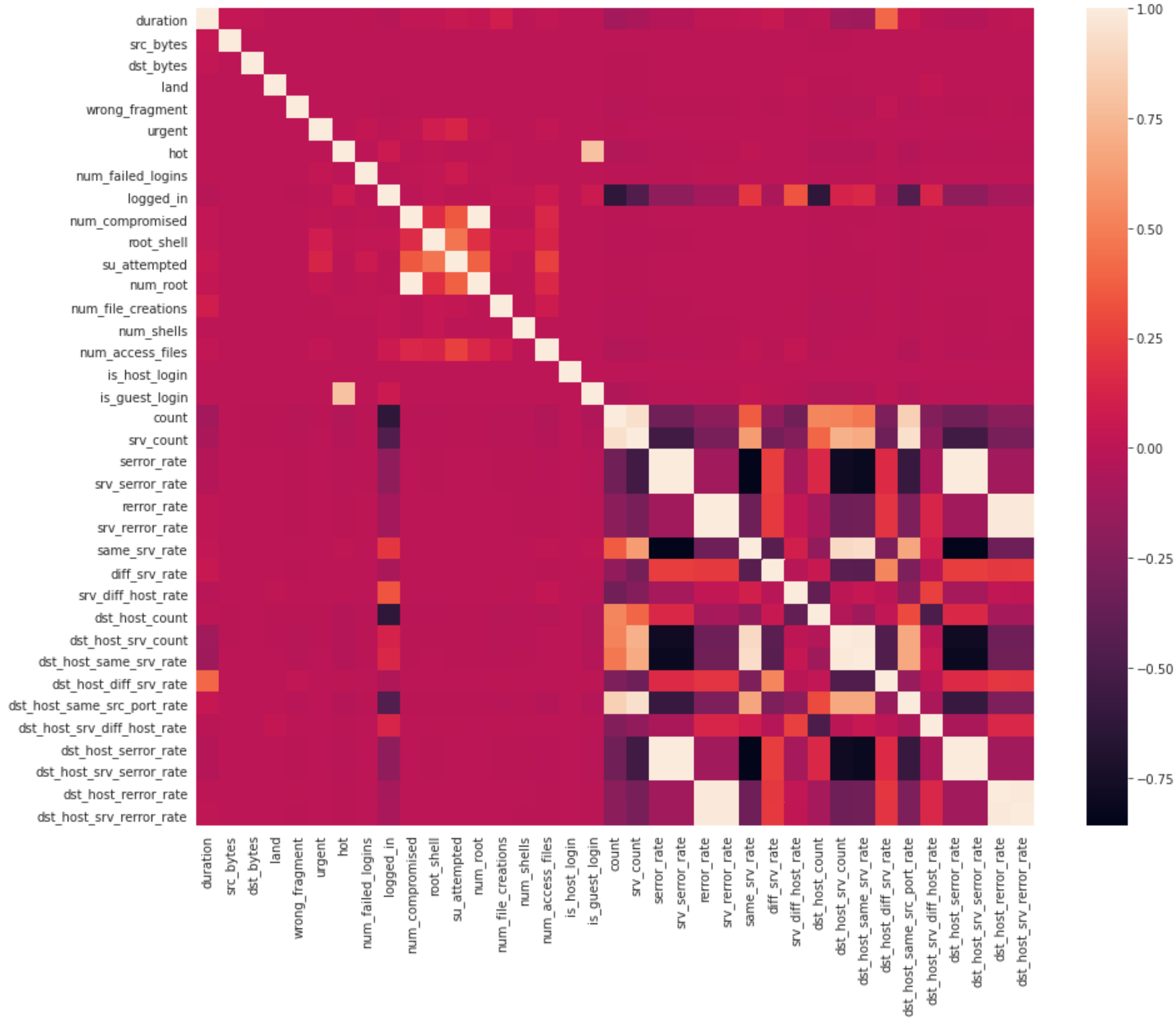
```
    41
```

Getting correlation matrix and plotting heatmap

```
correaltion_marix=training_df.corr()
plt.figure(figsize =(15, 12))

sns.heatmap(correaltion_marix)

plt.show()
```



Printing correlation matrix

```
correaltion_marix
```

num_access_files	0.023524	-2.225602e-05	0.000352	-0.000069	-0.(
is_host_login	-0.000021	-1.084695e-06	0.000004	-0.000002	-0.(
is_guest_login	0.002389	-3.608617e-05	0.000035	-0.000069	-0.(
count	-0.105074	-1.662632e-03	-0.002646	-0.003735	-0.(
srv_count	-0.079863	-1.150591e-03	-0.001998	-0.002852	-0.(
serror_rate	-0.031098	-5.858538e-04	-0.000774	0.004997	-0.(
srv_serror_rate	-0.031110	-6.321879e-04	-0.000773	0.005141	-0.(
rerror_rate	0.016549	3.209510e-03	0.002463	-0.000347	-0.(
srv_rerror_rate	0.016836	3.287307e-03	0.002467	-0.000593	-0.(
same_srv_rate	0.021719	6.696333e-04	0.000910	0.000926	0.(
diff_srv_rate	0.050286	3.294335e-04	-0.000393	0.000503	-0.(
srv_diff_host_rate	-0.012754	-1.422368e-04	0.000311	0.013491	0.(
dst_host_count	0.010914	-2.415847e-03	-0.001534	-0.008610	-0.(
dst_host_srv_count	-0.117309	-1.715221e-03	-0.001067	-0.004174	-0.(
dst_host_same_srv_rate	-0.119105	-1.548066e-03	-0.000968	0.000865	-0.(
dst_host_diff_srv_rate	0.409009	7.188089e-04	0.003307	-0.000236	0.(
dst_host_same_src_port_rate	0.042774	-7.931844e-04	-0.000558	0.001479	-0.(
dst_host_srv_diff_host_rate	-0.008582	4.755028e-06	0.000346	0.033193	0.(
dst_host_serror_rate	-0.030546	-8.206068e-04	-0.000765	0.004648	-0.(
dst_host_srv_serror_rate	-0.030570	-6.346845e-04	-0.000763	0.003096	-0.(
dst_host_rerror_rate	0.010569	-1.542303e-04	0.002502	-0.000552	0.(
dst_host_srv_rerror_rate	0.016034	2.927064e-03	0.002512	-0.000597	-0.(

Dropping following highly correlated features from dataset:

- num_root
- srv_serror_rate
- srv_rerror_rate
- dst_host_srv_serror_rate
- dst_host_serror_rate
- dst_host_rerror_rate
- dst_host_srv_rerror_rate
- dst_host_same_srv_rate

```
training_df.drop('num_root', axis = 1, inplace = True)
training_df.drop('srv_serror_rate', axis = 1, inplace = True)
training_df.drop('srv_rerror_rate', axis = 1, inplace = True)
training_df.drop('dst_host_srv_serror_rate', axis = 1, inplace = True)
training_df.drop('dst_host_serror_rate', axis = 1, inplace = True)
training_df.drop('dst_host_rerror_rate', axis = 1, inplace = True)
training_df.drop('dst_host_srv_rerror_rate', axis = 1, inplace = True)
training_df.drop('dst_host_same_srv_rate', axis = 1, inplace = True)


testing_df.drop('num_root', axis = 1, inplace = True)
testing_df.drop('srv_serror_rate', axis = 1, inplace = True)
testing_df.drop('srv_rerror_rate', axis = 1, inplace = True)
testing_df.drop('dst_host_srv_serror_rate', axis = 1, inplace = True)
testing_df.drop('dst_host_serror_rate', axis = 1, inplace = True)
testing_df.drop('dst_host_rerror_rate', axis = 1, inplace = True)
testing_df.drop('dst_host_srv_rerror_rate', axis = 1, inplace = True)
testing_df.drop('dst_host_same_srv_rate', axis = 1, inplace = True)
```

Printing length of current features after dropping

```
len(training_df.columns)
```

```
33
```

```
len(testing_df.columns)
```

```
32
```

Handling categorical data:

- protocol type feature mapping
- flag feature mapping

```
cat_map = {'protocol_map': {'icmp':0, 'tcp':1, 'udp':2}, 'flag_feature_map': {'SF'
training_df['protocol_type'] = training_df['protocol_type'].map(cat_map['protocol_
testing_df['protocol_type'] = testing_df['protocol_type'].map(cat_map['protocol_ma
training_df['flag'] = training_df['flag'].map(cat_map['flag_feature_map'])
testing_df['flag'] = testing_df['flag'].map(cat_map['flag_feature_map'])
```

```
training_df.drop('service', axis = 1, inplace = True)
testing_df.drop('service', axis = 1, inplace = True)
```

Extracting X_train, X_test and y_train

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
y=training_df["target"]
training_df.drop(['target', ], axis = 1,inplace=True)
```

```
testing_df=testing_df[1:]
```

Feature scaling

```
sc = MinMaxScaler()
X = sc.fit_transform(training_df)
X_test=sc.fit_transform(testing_df)
```

```
X_actual=np.copy(X)
y_actual=np.copy(y)
X_temp=np.copy(X)
y_temp=np.copy(y)
```

```
print(X_actual.shape)
print(y_actual.shape)

    (4898431, 31)
    (4898431,)
```

Training on Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clfg = GaussianNB()
start_time = time.time()
clfg.fit(X, y.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time, "seconds")

    Training time:  14.985315084457397 seconds
```

Printing accuracy, recall, precision, f1 score and other comparison parameters

```
y_train_pred_gauss = clfg.predict(X)
print('Accuracy:\n', accuracy_score(y, y_train_pred_gauss))
print('F1 score:\n', f1_score(y, y_train_pred_gauss, average= None))
print('Recall:\n', recall_score(y, y_train_pred_gauss, average= None))
print('Precision:\n', precision_score(y, y_train_pred_gauss, average= None))

    Accuracy:
     0.8695747679205852
    F1 score:
     [0.03341816 0.0028463  0.00108807 0.01206217 0.01022305 0.02992326
     0.85714286 0.01515152 0.03883495 0.9977684  0.02267832 0.53536328
     1.         1.         0.02895372 0.94211711 0.00115902 0.69589041
     0.99941702 0.8        0.37632135 0.00370588 0.0031343 ]
    Recall:
     [0.98955969 0.7        1.         0.98113208 0.91666667 0.06810352
     1.         0.66666667 0.28571429 0.99555231 0.72970639 0.36616361
     1.         1.         1.         0.9448766  0.5        0.95098163
     0.99883471 1.         1.         0.41372549 1.        ]
    Precision:
     [1.69960628e-02 1.42604916e-03 5.44328775e-04 6.06838604e-03
     5.14018692e-03 1.91739415e-02 7.50000000e-01 7.66283525e-03
     2.08333333e-02 9.99994378e-01 1.15181462e-02 9.95261153e-01
     1.00000000e+00 1.00000000e+00 1.46895170e-02 9.39373687e-01
```

```
5.80181016e-04 5.48705660e-01 1.00000000e+00 6.66666667e-01
2.31770833e-01 1.86127749e-03 1.56961231e-03]
```

```
print('\n clasification report:\n', classification_report(y, y_train_pred_gauss))
print('\n confussion matrix:\n',confusion_matrix(y, y_train_pred_gauss))
```

```
        0       0       0       0       0       0       0       0       0
        0       0       0       0       1]
 [      0       0       0       0      11       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       0       0       0       1]
 [      0       0      15      13       0     850       0       2       0
        0    3319       6       0       0    8248      27       0       0
        0       0       0       1       0]
 [      0       0       0       0       0       0      21       0       0
        0       0       0       0       0       0       0       0       0
        0       0       0       0       0]
 [      0       1       0       0       0       0       0       6       1
        0       0       0       0       0       0       0       0       0
        0       0       0       0       1]
 [      0       0       0       0       1       0       0       0       2
        0       0       0       0       0       0       0       0       0
        0       0       0       0       4]
 [      0       0       0       0      22       0       0       0       0
  1067249    2524      13       0       0       0     208       0    2001
        0       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0    1690       0       0       0     587       9      29       1
        0       0       0       0       0]
 [ 126085   14425   14515    8496    2099   43463       7     769      91
        6  137195  356197       0       0    8573     318    8545    9942
        0       1    3187  226294   12573]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       3       0       0       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       4       0       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0     264       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       1      18       0       0       0
        0      52       9       0       0       6    9839       0     486
        0       0       0       2       0]
 [      0       1       0       0       0       0       0       0       0
        0       2       0       0       0       0       0       5       0
        0       0       0       0       2]
 [      0       0       1       0       0       0       0       4       0
        0     561      25       0       0      36      73       7   15113
        0       0      58       6       8]
 [      0       0       0       0       0       0       0       0       0
        0    1382    1632       0       0     258       0       0       0
  2804614       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       2       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       0     979       0       0]
 [      0     277     157       1       0       0       0       2       0
        0       0       3       0       0       0       0      31       0
        0       0       0     422     127]
```

```
 [     0      0      0      0      0      0      0      0      0
       0      0      0      0      0      0      0      0      0
       0      0      0      0     20]]
```

Training on Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

clfd = DecisionTreeClassifier(criterion ="entropy", max_depth = 4)
start_time = time.time()
clfd.fit(X, y.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)
```

```
    Training time:  16.418840646743774
```

```
y_train_pred_dt = clfd.predict(X)
```

Printing accuracy, recall, precision, f1 score and other comparison parameters

```
print('Accuracy:', accuracy_score(y, y_train_pred_dt))
print('F1 score:', f1_score(y, y_train_pred_dt, average= None))
print('Recall:', recall_score(y, y_train_pred_dt, average= None))
print('Precision:', precision_score(y, y_train_pred_dt, average= None))
print('\n clasification report:\n', classification_report(y, y_train_pred_dt))
print('\n confussion matrix:\n',confusion_matrix(y, y_train_pred_gauss))
```

```
       0      0      0      0      0      0      0      0      0
       0      0      0      0      1]
 [     0      0      0      0     11      0      0      0      0
       0      0      0      0      0      0      0      0      0
       0      0      0      0      1]
 [     0      0     15     13      0    850      0      2      0
       0   3319      6      0      0   8248     27      0      0
       0      0      0      1      0]
 [     0      0      0      0      0      0     21      0      0
       0      0      0      0      0      0      0      0      0
       0      0      0      0      0]
 [     0      1      0      0      0      0      0      6      1
       0      0      0      0      0      0      0      0      0
       0      0      0      0      1]
 [     0      0      0      0      1      0      0      0      2
       0      0      0      0      0      0      0      0      0
       0      0      0      0      4]
 [     0      0      0      0     22      0      0      0      0
 1067249   2524     13      0      0      0    208      0   2001
       0      0      0      0      0]
 [     0      0      0      0      0      0      0      0      0
       0   1690      0      0      0    587      9     29      1
       0      0      0      0      0]
 [ 126085  14425  14515   8496   2099  43463      7    769     91
       6 137195 356197      0      0   8573    318   8545   9942
```

```
        0  137195  330197       0       0    8575     318    8545    9942
        0       1    3187  226294   12573]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       3       0       0       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       4       0       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0     264       0       0       0
        0       0       0       0       0]
 [      0       0       0       0       1      18       0       0       0
        0      52       9       0       0       6    9839       0     486
        0       0       0       2       0]
 [      0       1       0       0       0       0       0       0       0
        0       2       0       0       0       0       0       5       0
        0       0       0       0       2]
 [      0       0       1       0       0       0       0       4       0
        0     561      25       0       0      36      73       7   15113
        0       0      58       6       8]
 [      0       0       0       0       0       0       0       0       0
        0    1382    1632       0       0     258       0       0       0
  2804614       0       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       2       0       0       0]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       0     979       0       0]
 [      0     277     157       1       0       0       0       2       0
        0       0       3       0       0       0       0      31       0
        0       0       0     422     127]
 [      0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0       0
        0       0       0       0     201]
```

Training on random forest

```
from sklearn.ensemble import RandomForestClassifier

clfr = RandomForestClassifier(n_estimators = 30)
start_time = time.time()
clfr.fit(X, y.values.ravel())
end_time = time.time()
print("Training time: ", end_time-start_time)
```

```
Training time:  135.25770211219788
```

```
y_train_pred_rf = clfr.predict(X)
```

Printing accuracy, recall, precision, f1 score and other comparison parameters

```
print('Accuracy:', accuracy_score(y, y_train_pred_rf))
```

```
print('Accuracy:', accuracy_score(y, y_train_pred_rf))
print('F1 score:', f1_score(y, y_train_pred_rf, average= None))
print('Recall:', recall_score(y, y_train_pred_rf, average= None))
print('Precision:', precision_score(y, y_train_pred_rf, average= None))
print('\n clasification report:\n', classification_report(y, y_train_pred_rf))
print('\n confussion matrix:\n',confusion_matrix(y, y_train_pred_rf))
```

```
 [       0       0       0      53       0       0       0       0       0
         0       0       0       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0      12       0       0       0       0
         0       0       0       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0   12411       0       0       0
         0       2      68       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0      21       0       0
         0       0       0       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       9       0
         0       0       0       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       6
         0       0       1       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
   1072017       0       0       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0      20       0       0       0
         0    2242      54       0       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       4       5       0       0
         0       0  972760       0       0       1       1       0       0
         0       0       0      10       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       0       3       0       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       0       0       4       0       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       2       0       0     262       0       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       3       0       0       0   10410       0       0
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       3       0       0       0       0       7       0
         0       0       0       0       0]
 [       0       0       0       0       0       2       0       0       0
         0       0      35       0       0       0       0       0   15855
         0       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       1       0       0       0       0       0       0
   2807885       0       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       0       0       0       0       0       0       0
         0       2       0       0       0]
 [       0       0       0       0       0       0       0       0       0
         0       0       0       0       0       0       0       0       0
         0       0     979       0       0]
 [       0       0       0       0       0       0       0       0       0
```

```
         0        0        1        0        0        0        0        0        0
         0        0        0     1019        0]
 [       0        0        0        0        0        0        0        0        0
```

Printing accuracy, recall, precision, f1 score and other comparison parameters

From above, we saw that the best performance on training data was given by tree based models. Selecting random forest as our model, we perform the following:

- predict label of test data
- store them in a csv file

```
start_time = time.time()
y_test_pred = clfr.predict(X_test)
end_time = time.time()
print("Execution time:", end_time - start_time, " seconds")
print(y_test_pred)
print(y_test_pred.shape)
print(type(y_test_pred))
df_ans = pd.DataFrame(y_test_pred, columns = ['target'])
print(df_ans)
df_ans.to_csv('testLabel.csv', header="True", index= "False")
```

✓ 7m 29s completed at 12:33