In [1]:

```
# Part 1: Understanding the data #

print ('What is the objective of the data collection process?')

print ('--To collect the useful data as sampling that illustrate real world ph
ysical conditions based on the research or project objective, which these data
can then convert into digital numeric values that allow the computer to proces
s and manipulate, subsequently to be analysed so that the input data can be us
ed to design an accurate model in Machine Learning.')
```

```
What is the objective of the data collection process?
--To collect the useful data as sampling that illustrate real worl
d physical conditions based on the research or project objective,
which these data can then convert into digital numeric values that
allow the computer to process and manipulate, subsequently to be a
nalysed so that the input data can be used to design an accurate m
odel in Machine Learning.
```

In [35]:

```
print ('What human activity types does this dataset have? How many subjects/pe
ople have performed these activities?')

print ('--Human activity types for this dataset : (i) Stand, (ii) Sit, (iii) L
ay Down, (iv) Walk, (v) Walk Downstairs and (vi) Walk Upstairs')

print ('--Number of subject / people had performed these activities : 30')
```

```
What human activity types does this dataset have? How many subject
s/people have performed these activities?
--Human activity types for this dataset : (i) Stand, (ii) Sit, (ii
i) Lay Down, (iv) Walk, (v) Walk Downstairs and (vi) Walk Upstairs
--Number of subject / people had performed these activities : 30
```

In [34]:

```
print ('How many instances are available in the training and test sets? How ma
ny features are used to represent each instance? Summarize the type of feature
s extracted in 2-3 sentences.')

print ('--Number of instances available in the training sets = 7352')

print ('--Number of instances available in the test sets = 2947')

print ('--Number of features used to represent each instance = 561')

print ('--Type of features extracted : Including mean, min, max, median, stand
ard deviation, correlation, signal magnitude area, Average sum of the squares,
Interquartile range, Signal Entropy, Autorregresion coefficients, Correlation
coefficient, Largest frequency component, Frequency signal weighted average, F
requency signal Skewness, Frequency signal Kurtosis, Energy of a frequency int
erval and Angle between two vectors. ')
```

```
How many instances are available in the training and test sets? Ho
w many features are used to represent each instance? Summarize the
type of features extracted in 2-3 sentences.
--Number of instances available in the training sets = 7352
--Number of instances available in the test sets = 2947
--Number of features used to represent each instance = 561
--Type of features extracted : Including mean, min, max, median, s
tandard deviation, correlation, signal magnitude area, Average sum
of the squares, Interquartile range, Signal Entropy, Autorregresio
n coefficients, Correlation coefficient, Largest frequency compone
nt, Frequency signal weighted average, Frequency signal Skewness,
Frequency signal Kurtosis, Energy of a frequency interval and Angl
e between two vectors.
```

In [4]:

```
print ('Describe briefly what machine learning model is used in this paper for activity recognition and how is it trained. How much is the maximum accuracy achieved?')

print ('--Machine learning model used in this paper : Supervised Learning Classification Model')

print ('--70% of the data input was used to trained the model, the relationship between the input and output are known.The Support Vector Machine hyperparameters are selected through a 10-fold Cross Validation procedure and Gaussian kernels are used.')

print ('--The maximum accuracy achieved is 96%')
```

```
Describe briefly what machine learning model is used in this paper
for activity recognition and how is it trained. How much is the ma
ximum accuracy achieved?
--Machine learning model used in this paper : Supervised Learning
Classification Model
--70% of the data input was used to trained the model, the relatio
nship between the input and output are known.The Support Vector Ma
chine hyperparameters are selected through a 10-fold Cross Validat
ion procedure and Gaussian kernels are used.
--The maximum accuracy achieved is 96%
```

In [5]:

```
# Part 2: K-Nearest Neighbour Classification #

import numpy as np
import pandas as pd
import csv

def gen_rows(stream, max_length=None):
      rows = csv.reader(stream)
      if max_length is None:
          rows = list(rows)
          max_length = max(len(row) for row in rows)
      for row in rows:
          yield row + [None] * (max_length - len(row))

def get_csv_from_txt(filename):
    with open(filename, 'r') as in_file:
        stripped = (line.strip() for line in in_file)
        lines = (line.split(" ") for line in stripped if line)
    #print (lines)
        with open('temporary.csv', 'w') as out_file:
            writer = csv.writer(out_file)
            writer.writerows(lines)
    with open('temporary.csv') as f:
        d1 = pd.DataFrame.from_records(list(gen_rows(f)))
        return d1
```

In [6]:

```
def read_train_data():
    cols = 240
    filenames = ['X_train.txt']
    Train_Data = get_csv_from_txt('/Users/limshikee/Desktop/train/' + filename
s[0])
    Train_Data = Train_Data[np.arange(cols)]
    filesl = len(filenames)
    i = 1
    while i<filesl:
        s_train = get_csv_from_txt('/Users/limshikee/Desktop/train/' + filenam
es[i])
        s_train = s_train[np.arange(cols)]
        Train_Data = pd.concat([Train_Data, s_train], axis=1)
        i+=1
    ytrain = get_csv_from_txt('/Users/limshikee/Desktop/train/y_train.txt')
    return Train_Data,ytrain

def read_test_data():
    filenames = ['X_test.txt']
    cols = 240
    Test_Data = get_csv_from_txt('/Users/limshikee/Desktop/test/' + filenames[
0])
    Test_Data = Test_Data[np.arange(cols)]
    filesl = len(filenames)
    i = 1
    while i<filesl:
        s_test = get_csv_from_txt('/Users/limshikee/Desktop/test/' + filenames
[i])
        s_test = s_test[np.arange(cols)]
        Test_Data = pd.concat([Test_Data, s_test], axis=1)
        i+=1
    ytest = get_csv_from_txt('/Users/limshikee/Desktop/test/y_test.txt')
    return Test_Data,ytest
```

In [7]:

```
X_Train,Y_Train = read_train_data()
X_Test,Y_Test = read_test_data()
```

In [36]:

```
from sklearn.preprocessing import Imputer

X_Train = X_Train.replace('', 'Nan')
X_Train = X_Train.astype(float)

X_Test = X_Test.replace('', 'Nan')
X_Test = X_Test.astype(float)
```

In [9]:

```
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(X_Train)
X_Train = pd.DataFrame(imp.transform(X_Train))
X_Test = pd.DataFrame(imp.transform(X_Test))
```

```
/Users/limshikee/anaconda3/lib/python3.7/site-packages/sklearn/uti
ls/deprecation.py:58: DeprecationWarning: Class Imputer is depreca
ted; Imputer was deprecated in version 0.20 and will be removed in
0.22. Import impute.SimpleImputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

In [10]:

```
print (len(X_Train))
print (len(Y_Train))

print (len(X_Test))
print (len(Y_Test))
```

```
7352
7352
2947
2947
```

In [11]:

```
fID = 218187754%2

fID
```

Out[11]:

```
0
```

In [12]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

In [37]:

```
# Cross Validation
from sklearn.model_selection import train_test_split
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

acc = []
for k in np.arange(1,51):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = []
    for i in np.arange(1,11):
        #Xtrain, Xtest, ytrain, ytest = train_test_split(X_Train, Y_Train)
        knn.fit(X_Train,np.array(Y_Train))
        pred = knn.predict(X_Test)
        f1 = f1_score(np.array(Y_Test),pred,average='micro')
        print ('k:',k,' k_fold:',i,' f1_score:',f1)
        scores.append(f1)
    macc = np.mean(scores)
    print ('k:',k,' Mean Acc:',macc)
    acc.append(macc)
```

```
k: 1  k_fold: 1  f1_score: 0.6314896504920258
k: 1  k_fold: 2  f1_score: 0.6314896504920258
k: 1  k_fold: 3  f1_score: 0.6314896504920258
k: 1  k_fold: 4  f1_score: 0.6314896504920258
k: 1  k_fold: 5  f1_score: 0.6314896504920258
k: 1  k_fold: 6  f1_score: 0.6314896504920258
k: 1  k_fold: 7  f1_score: 0.6314896504920258
k: 1  k_fold: 8  f1_score: 0.6314896504920258
k: 1  k_fold: 9  f1_score: 0.6314896504920258
k: 1  k_fold: 10  f1_score: 0.6314896504920258
k: 1  Mean Acc: 0.6314896504920258
k: 2  k_fold: 1  f1_score: 0.5802511028164234
k: 2  k_fold: 2  f1_score: 0.5802511028164234
k: 2  k_fold: 3  f1_score: 0.5802511028164234
k: 2  k_fold: 4  f1_score: 0.5802511028164234
k: 2  k_fold: 5  f1_score: 0.5802511028164234
k: 2  k_fold: 6  f1_score: 0.5802511028164234
k: 2  k_fold: 7  f1_score: 0.5802511028164234
k: 2  k_fold: 8  f1_score: 0.5802511028164234
k: 2  k_fold: 9  f1_score: 0.5802511028164234
k: 2  k_fold: 10  f1_score: 0.5802511028164234
k: 2  Mean Acc: 0.5802511028164234
k: 3  k_fold: 1  f1_score: 0.6308109942314218
k: 3  k_fold: 2  f1_score: 0.6308109942314218
k: 3  k_fold: 3  f1_score: 0.6308109942314218
k: 3  k_fold: 4  f1_score: 0.6308109942314218
k: 3  k_fold: 5  f1_score: 0.6308109942314218
k: 3  k_fold: 6  f1_score: 0.6308109942314218
k: 3  k_fold: 7  f1_score: 0.6308109942314218
k: 3  k_fold: 8  f1_score: 0.6308109942314218
k: 3  k_fold: 9  f1_score: 0.6308109942314218
k: 3  k_fold: 10  f1_score: 0.6308109942314218
k: 3  Mean Acc: 0.6308109942314218
```

```
k: 4  k_fold: 1  f1_score: 0.6223277909738717
k: 4  k_fold: 2  f1_score: 0.6223277909738717
k: 4  k_fold: 3  f1_score: 0.6223277909738717
k: 4  k_fold: 4  f1_score: 0.6223277909738717
k: 4  k_fold: 5  f1_score: 0.6223277909738717
k: 4  k_fold: 6  f1_score: 0.6223277909738717
k: 4  k_fold: 7  f1_score: 0.6223277909738717
k: 4  k_fold: 8  f1_score: 0.6223277909738717
k: 4  k_fold: 9  f1_score: 0.6223277909738717
k: 4  k_fold: 10  f1_score: 0.6223277909738717
k: 4  Mean Acc: 0.6223277909738717
k: 5  k_fold: 1  f1_score: 0.6342042755344418
k: 5  k_fold: 2  f1_score: 0.6342042755344418
k: 5  k_fold: 3  f1_score: 0.6342042755344418
k: 5  k_fold: 4  f1_score: 0.6342042755344418
k: 5  k_fold: 5  f1_score: 0.6342042755344418
k: 5  k_fold: 6  f1_score: 0.6342042755344418
k: 5  k_fold: 7  f1_score: 0.6342042755344418
k: 5  k_fold: 8  f1_score: 0.6342042755344418
k: 5  k_fold: 9  f1_score: 0.6342042755344418
k: 5  k_fold: 10  f1_score: 0.6342042755344418
k: 5  Mean Acc: 0.6342042755344418
k: 6  k_fold: 1  f1_score: 0.6338649474041398
k: 6  k_fold: 2  f1_score: 0.6338649474041398
k: 6  k_fold: 3  f1_score: 0.6338649474041398
k: 6  k_fold: 4  f1_score: 0.6338649474041398
k: 6  k_fold: 5  f1_score: 0.6338649474041398
k: 6  k_fold: 6  f1_score: 0.6338649474041398
k: 6  k_fold: 7  f1_score: 0.6338649474041398
k: 6  k_fold: 8  f1_score: 0.6338649474041398
k: 6  k_fold: 9  f1_score: 0.6338649474041398
k: 6  k_fold: 10  f1_score: 0.6338649474041398
k: 6  Mean Acc: 0.6338649474041398
k: 7  k_fold: 1  f1_score: 0.6416694944010859
k: 7  k_fold: 2  f1_score: 0.6416694944010859
k: 7  k_fold: 3  f1_score: 0.6416694944010859
k: 7  k_fold: 4  f1_score: 0.6416694944010859
k: 7  k_fold: 5  f1_score: 0.6416694944010859
k: 7  k_fold: 6  f1_score: 0.6416694944010859
k: 7  k_fold: 7  f1_score: 0.6416694944010859
k: 7  k_fold: 8  f1_score: 0.6416694944010859
k: 7  k_fold: 9  f1_score: 0.6416694944010859
k: 7  k_fold: 10  f1_score: 0.6416694944010859
k: 7  Mean Acc: 0.6416694944010859
k: 8  k_fold: 1  f1_score: 0.6379368849677638
k: 8  k_fold: 2  f1_score: 0.6379368849677638
k: 8  k_fold: 3  f1_score: 0.6379368849677638
k: 8  k_fold: 4  f1_score: 0.6379368849677638
k: 8  k_fold: 5  f1_score: 0.6379368849677638
k: 8  k_fold: 6  f1_score: 0.6379368849677638
k: 8  k_fold: 7  f1_score: 0.6379368849677638
k: 8  k_fold: 8  f1_score: 0.6379368849677638
k: 8  k_fold: 9  f1_score: 0.6379368849677638
k: 8  k_fold: 10  f1_score: 0.6379368849677638
k: 8  Mean Acc: 0.6379368849677638
k: 9  k_fold: 1  f1_score: 0.6443841194435018
k: 9  k_fold: 2  f1_score: 0.6443841194435018
```

```
k: 9  k_fold: 3  f1_score: 0.6443841194435018
k: 9  k_fold: 4  f1_score: 0.6443841194435018
k: 9  k_fold: 5  f1_score: 0.6443841194435018
k: 9  k_fold: 6  f1_score: 0.6443841194435018
k: 9  k_fold: 7  f1_score: 0.6443841194435018
k: 9  k_fold: 8  f1_score: 0.6443841194435018
k: 9  k_fold: 9  f1_score: 0.6443841194435018
k: 9  k_fold: 10  f1_score: 0.6443841194435018
k: 9  Mean Acc: 0.6443841194435018
k: 10  k_fold: 1  f1_score: 0.6403121818798778
k: 10  k_fold: 2  f1_score: 0.6403121818798778
k: 10  k_fold: 3  f1_score: 0.6403121818798778
k: 10  k_fold: 4  f1_score: 0.6403121818798778
k: 10  k_fold: 5  f1_score: 0.6403121818798778
k: 10  k_fold: 6  f1_score: 0.6403121818798778
k: 10  k_fold: 7  f1_score: 0.6403121818798778
k: 10  k_fold: 8  f1_score: 0.6403121818798778
k: 10  k_fold: 9  f1_score: 0.6403121818798778
k: 10  k_fold: 10  f1_score: 0.6403121818798778
k: 10  Mean Acc: 0.6403121818798778
k: 11  k_fold: 1  f1_score: 0.6447234475738038
k: 11  k_fold: 2  f1_score: 0.6447234475738038
k: 11  k_fold: 3  f1_score: 0.6447234475738038
k: 11  k_fold: 4  f1_score: 0.6447234475738038
k: 11  k_fold: 5  f1_score: 0.6447234475738038
k: 11  k_fold: 6  f1_score: 0.6447234475738038
k: 11  k_fold: 7  f1_score: 0.6447234475738038
k: 11  k_fold: 8  f1_score: 0.6447234475738038
k: 11  k_fold: 9  f1_score: 0.6447234475738038
k: 11  k_fold: 10  f1_score: 0.6447234475738038
k: 11  Mean Acc: 0.6447234475738038
k: 12  k_fold: 1  f1_score: 0.6450627757041059
k: 12  k_fold: 2  f1_score: 0.6450627757041059
k: 12  k_fold: 3  f1_score: 0.6450627757041059
k: 12  k_fold: 4  f1_score: 0.6450627757041059
k: 12  k_fold: 5  f1_score: 0.6450627757041059
k: 12  k_fold: 6  f1_score: 0.6450627757041059
k: 12  k_fold: 7  f1_score: 0.6450627757041059
k: 12  k_fold: 8  f1_score: 0.6450627757041059
k: 12  k_fold: 9  f1_score: 0.6450627757041059
k: 12  k_fold: 10  f1_score: 0.6450627757041059
k: 12  Mean Acc: 0.6450627757041059
k: 13  k_fold: 1  f1_score: 0.6484560570071259
k: 13  k_fold: 2  f1_score: 0.6484560570071259
k: 13  k_fold: 3  f1_score: 0.6484560570071259
k: 13  k_fold: 4  f1_score: 0.6484560570071259
k: 13  k_fold: 5  f1_score: 0.6484560570071259
k: 13  k_fold: 6  f1_score: 0.6484560570071259
k: 13  k_fold: 7  f1_score: 0.6484560570071259
k: 13  k_fold: 8  f1_score: 0.6484560570071259
k: 13  k_fold: 9  f1_score: 0.6484560570071259
k: 13  k_fold: 10  f1_score: 0.6484560570071259
k: 13  Mean Acc: 0.6484560570071258
k: 14  k_fold: 1  f1_score: 0.6481167288768239
k: 14  k_fold: 2  f1_score: 0.6481167288768239
k: 14  k_fold: 3  f1_score: 0.6481167288768239
k: 14  k_fold: 4  f1_score: 0.6481167288768239
```

```
k: 14  k_fold: 5  f1_score: 0.6481167288768239
k: 14  k_fold: 6  f1_score: 0.6481167288768239
k: 14  k_fold: 7  f1_score: 0.6481167288768239
k: 14  k_fold: 8  f1_score: 0.6481167288768239
k: 14  k_fold: 9  f1_score: 0.6481167288768239
k: 14  k_fold: 10  f1_score: 0.6481167288768239
k: 14  Mean Acc: 0.6481167288768239
k: 15  k_fold: 1  f1_score: 0.6454021038344079
k: 15  k_fold: 2  f1_score: 0.6454021038344079
k: 15  k_fold: 3  f1_score: 0.6454021038344079
k: 15  k_fold: 4  f1_score: 0.6454021038344079
k: 15  k_fold: 5  f1_score: 0.6454021038344079
k: 15  k_fold: 6  f1_score: 0.6454021038344079
k: 15  k_fold: 7  f1_score: 0.6454021038344079
k: 15  k_fold: 8  f1_score: 0.6454021038344079
k: 15  k_fold: 9  f1_score: 0.6454021038344079
k: 15  k_fold: 10  f1_score: 0.6454021038344079
k: 15  Mean Acc: 0.6454021038344078
k: 16  k_fold: 1  f1_score: 0.6443841194435018
k: 16  k_fold: 2  f1_score: 0.6443841194435018
k: 16  k_fold: 3  f1_score: 0.6443841194435018
k: 16  k_fold: 4  f1_score: 0.6443841194435018
k: 16  k_fold: 5  f1_score: 0.6443841194435018
k: 16  k_fold: 6  f1_score: 0.6443841194435018
k: 16  k_fold: 7  f1_score: 0.6443841194435018
k: 16  k_fold: 8  f1_score: 0.6443841194435018
k: 16  k_fold: 9  f1_score: 0.6443841194435018
k: 16  k_fold: 10  f1_score: 0.6443841194435018
k: 16  Mean Acc: 0.6443841194435018
k: 17  k_fold: 1  f1_score: 0.6392941974889719
k: 17  k_fold: 2  f1_score: 0.6392941974889719
k: 17  k_fold: 3  f1_score: 0.6392941974889719
k: 17  k_fold: 4  f1_score: 0.6392941974889719
k: 17  k_fold: 5  f1_score: 0.6392941974889719
k: 17  k_fold: 6  f1_score: 0.6392941974889719
k: 17  k_fold: 7  f1_score: 0.6392941974889719
k: 17  k_fold: 8  f1_score: 0.6392941974889719
k: 17  k_fold: 9  f1_score: 0.6392941974889719
k: 17  k_fold: 10  f1_score: 0.6392941974889719
k: 17  Mean Acc: 0.639294197488972
k: 18  k_fold: 1  f1_score: 0.6338649474041398
k: 18  k_fold: 2  f1_score: 0.6338649474041398
k: 18  k_fold: 3  f1_score: 0.6338649474041398
k: 18  k_fold: 4  f1_score: 0.6338649474041398
k: 18  k_fold: 5  f1_score: 0.6338649474041398
k: 18  k_fold: 6  f1_score: 0.6338649474041398
k: 18  k_fold: 7  f1_score: 0.6338649474041398
k: 18  k_fold: 8  f1_score: 0.6338649474041398
k: 18  k_fold: 9  f1_score: 0.6338649474041398
k: 18  k_fold: 10  f1_score: 0.6338649474041398
k: 18  Mean Acc: 0.6338649474041398
k: 19  k_fold: 1  f1_score: 0.6362402443162538
k: 19  k_fold: 2  f1_score: 0.6362402443162538
k: 19  k_fold: 3  f1_score: 0.6362402443162538
k: 19  k_fold: 4  f1_score: 0.6362402443162538
k: 19  k_fold: 5  f1_score: 0.6362402443162538
k: 19  k_fold: 6  f1_score: 0.6362402443162538
```

```
k: 19  k_fold: 7  f1_score: 0.6362402443162538
k: 19  k_fold: 8  f1_score: 0.6362402443162538
k: 19  k_fold: 9  f1_score: 0.6362402443162538
k: 19  k_fold: 10  f1_score: 0.6362402443162538
k: 19  Mean Acc: 0.6362402443162537
k: 20  k_fold: 1  f1_score: 0.6331862911435358
k: 20  k_fold: 2  f1_score: 0.6331862911435358
k: 20  k_fold: 3  f1_score: 0.6331862911435358
k: 20  k_fold: 4  f1_score: 0.6331862911435358
k: 20  k_fold: 5  f1_score: 0.6331862911435358
k: 20  k_fold: 6  f1_score: 0.6331862911435358
k: 20  k_fold: 7  f1_score: 0.6331862911435358
k: 20  k_fold: 8  f1_score: 0.6331862911435358
k: 20  k_fold: 9  f1_score: 0.6331862911435358
k: 20  k_fold: 10  f1_score: 0.6331862911435358
k: 20  Mean Acc: 0.6331862911435359
k: 21  k_fold: 1  f1_score: 0.6338649474041398
k: 21  k_fold: 2  f1_score: 0.6338649474041398
k: 21  k_fold: 3  f1_score: 0.6338649474041398
k: 21  k_fold: 4  f1_score: 0.6338649474041398
k: 21  k_fold: 5  f1_score: 0.6338649474041398
k: 21  k_fold: 6  f1_score: 0.6338649474041398
k: 21  k_fold: 7  f1_score: 0.6338649474041398
k: 21  k_fold: 8  f1_score: 0.6338649474041398
k: 21  k_fold: 9  f1_score: 0.6338649474041398
k: 21  k_fold: 10  f1_score: 0.6338649474041398
k: 21  Mean Acc: 0.6338649474041398
k: 22  k_fold: 1  f1_score: 0.6359009161859518
k: 22  k_fold: 2  f1_score: 0.6359009161859518
k: 22  k_fold: 3  f1_score: 0.6359009161859518
k: 22  k_fold: 4  f1_score: 0.6359009161859518
k: 22  k_fold: 5  f1_score: 0.6359009161859518
k: 22  k_fold: 6  f1_score: 0.6359009161859518
k: 22  k_fold: 7  f1_score: 0.6359009161859518
k: 22  k_fold: 8  f1_score: 0.6359009161859518
k: 22  k_fold: 9  f1_score: 0.6359009161859518
k: 22  k_fold: 10  f1_score: 0.6359009161859518
k: 22  Mean Acc: 0.6359009161859517
k: 23  k_fold: 1  f1_score: 0.6311503223617237
k: 23  k_fold: 2  f1_score: 0.6311503223617237
k: 23  k_fold: 3  f1_score: 0.6311503223617237
k: 23  k_fold: 4  f1_score: 0.6311503223617237
k: 23  k_fold: 5  f1_score: 0.6311503223617237
k: 23  k_fold: 6  f1_score: 0.6311503223617237
k: 23  k_fold: 7  f1_score: 0.6311503223617237
k: 23  k_fold: 8  f1_score: 0.6311503223617237
k: 23  k_fold: 9  f1_score: 0.6311503223617237
k: 23  k_fold: 10  f1_score: 0.6311503223617237
k: 23  Mean Acc: 0.6311503223617237
k: 24  k_fold: 1  f1_score: 0.6335256192738378
k: 24  k_fold: 2  f1_score: 0.6335256192738378
k: 24  k_fold: 3  f1_score: 0.6335256192738378
k: 24  k_fold: 4  f1_score: 0.6335256192738378
k: 24  k_fold: 5  f1_score: 0.6335256192738378
k: 24  k_fold: 6  f1_score: 0.6335256192738378
k: 24  k_fold: 7  f1_score: 0.6335256192738378
k: 24  k_fold: 8  f1_score: 0.6335256192738378
```

```
k: 24  k_fold: 9  f1_score: 0.6335256192738378
k: 24  k_fold: 10  f1_score: 0.6335256192738378
k: 24  Mean Acc: 0.6335256192738379
k: 25  k_fold: 1  f1_score: 0.6304716661011198
k: 25  k_fold: 2  f1_score: 0.6304716661011198
k: 25  k_fold: 3  f1_score: 0.6304716661011198
k: 25  k_fold: 4  f1_score: 0.6304716661011198
k: 25  k_fold: 5  f1_score: 0.6304716661011198
k: 25  k_fold: 6  f1_score: 0.6304716661011198
k: 25  k_fold: 7  f1_score: 0.6304716661011198
k: 25  k_fold: 8  f1_score: 0.6304716661011198
k: 25  k_fold: 9  f1_score: 0.6304716661011198
k: 25  k_fold: 10  f1_score: 0.6304716661011198
k: 25  Mean Acc: 0.6304716661011198
k: 26  k_fold: 1  f1_score: 0.6314896504920258
k: 26  k_fold: 2  f1_score: 0.6314896504920258
k: 26  k_fold: 3  f1_score: 0.6314896504920258
k: 26  k_fold: 4  f1_score: 0.6314896504920258
k: 26  k_fold: 5  f1_score: 0.6314896504920258
k: 26  k_fold: 6  f1_score: 0.6314896504920258
k: 26  k_fold: 7  f1_score: 0.6314896504920258
k: 26  k_fold: 8  f1_score: 0.6314896504920258
k: 26  k_fold: 9  f1_score: 0.6314896504920258
k: 26  k_fold: 10  f1_score: 0.6314896504920258
k: 26  Mean Acc: 0.6314896504920258
k: 27  k_fold: 1  f1_score: 0.6314896504920258
k: 27  k_fold: 2  f1_score: 0.6314896504920258
k: 27  k_fold: 3  f1_score: 0.6314896504920258
k: 27  k_fold: 4  f1_score: 0.6314896504920258
k: 27  k_fold: 5  f1_score: 0.6314896504920258
k: 27  k_fold: 6  f1_score: 0.6314896504920258
k: 27  k_fold: 7  f1_score: 0.6314896504920258
k: 27  k_fold: 8  f1_score: 0.6314896504920258
k: 27  k_fold: 9  f1_score: 0.6314896504920258
k: 27  k_fold: 10  f1_score: 0.6314896504920258
k: 27  Mean Acc: 0.6314896504920258
k: 28  k_fold: 1  f1_score: 0.6284356973193078
k: 28  k_fold: 2  f1_score: 0.6284356973193078
k: 28  k_fold: 3  f1_score: 0.6284356973193078
k: 28  k_fold: 4  f1_score: 0.6284356973193078
k: 28  k_fold: 5  f1_score: 0.6284356973193078
k: 28  k_fold: 6  f1_score: 0.6284356973193078
k: 28  k_fold: 7  f1_score: 0.6284356973193078
k: 28  k_fold: 8  f1_score: 0.6284356973193078
k: 28  k_fold: 9  f1_score: 0.6284356973193078
k: 28  k_fold: 10  f1_score: 0.6284356973193078
k: 28  Mean Acc: 0.6284356973193078
k: 29  k_fold: 1  f1_score: 0.6267390566677977
k: 29  k_fold: 2  f1_score: 0.6267390566677977
k: 29  k_fold: 3  f1_score: 0.6267390566677977
k: 29  k_fold: 4  f1_score: 0.6267390566677977
k: 29  k_fold: 5  f1_score: 0.6267390566677977
k: 29  k_fold: 6  f1_score: 0.6267390566677977
k: 29  k_fold: 7  f1_score: 0.6267390566677977
k: 29  k_fold: 8  f1_score: 0.6267390566677977
k: 29  k_fold: 9  f1_score: 0.6267390566677977
k: 29  k_fold: 10  f1_score: 0.6267390566677977
```

```
k: 29  Mean Acc: 0.6267390566677978
k: 30  k_fold: 1  f1_score: 0.6257210722768918
k: 30  k_fold: 2  f1_score: 0.6257210722768918
k: 30  k_fold: 3  f1_score: 0.6257210722768918
k: 30  k_fold: 4  f1_score: 0.6257210722768918
k: 30  k_fold: 5  f1_score: 0.6257210722768918
k: 30  k_fold: 6  f1_score: 0.6257210722768918
k: 30  k_fold: 7  f1_score: 0.6257210722768918
k: 30  k_fold: 8  f1_score: 0.6257210722768918
k: 30  k_fold: 9  f1_score: 0.6257210722768918
k: 30  k_fold: 10  f1_score: 0.6257210722768918
k: 30  Mean Acc: 0.6257210722768918
k: 31  k_fold: 1  f1_score: 0.6257210722768918
k: 31  k_fold: 2  f1_score: 0.6257210722768918
k: 31  k_fold: 3  f1_score: 0.6257210722768918
k: 31  k_fold: 4  f1_score: 0.6257210722768918
k: 31  k_fold: 5  f1_score: 0.6257210722768918
k: 31  k_fold: 6  f1_score: 0.6257210722768918
k: 31  k_fold: 7  f1_score: 0.6257210722768918
k: 31  k_fold: 8  f1_score: 0.6257210722768918
k: 31  k_fold: 9  f1_score: 0.6257210722768918
k: 31  k_fold: 10  f1_score: 0.6257210722768918
k: 31  Mean Acc: 0.6257210722768918
k: 32  k_fold: 1  f1_score: 0.6233457753647778
k: 32  k_fold: 2  f1_score: 0.6233457753647778
k: 32  k_fold: 3  f1_score: 0.6233457753647778
k: 32  k_fold: 4  f1_score: 0.6233457753647778
k: 32  k_fold: 5  f1_score: 0.6233457753647778
k: 32  k_fold: 6  f1_score: 0.6233457753647778
k: 32  k_fold: 7  f1_score: 0.6233457753647778
k: 32  k_fold: 8  f1_score: 0.6233457753647778
k: 32  k_fold: 9  f1_score: 0.6233457753647778
k: 32  k_fold: 10  f1_score: 0.6233457753647778
k: 32  Mean Acc: 0.6233457753647776
k: 33  k_fold: 1  f1_score: 0.6270783847980997
k: 33  k_fold: 2  f1_score: 0.6270783847980997
k: 33  k_fold: 3  f1_score: 0.6270783847980997
k: 33  k_fold: 4  f1_score: 0.6270783847980997
k: 33  k_fold: 5  f1_score: 0.6270783847980997
k: 33  k_fold: 6  f1_score: 0.6270783847980997
k: 33  k_fold: 7  f1_score: 0.6270783847980997
k: 33  k_fold: 8  f1_score: 0.6270783847980997
k: 33  k_fold: 9  f1_score: 0.6270783847980997
k: 33  k_fold: 10  f1_score: 0.6270783847980997
k: 33  Mean Acc: 0.6270783847980999
k: 34  k_fold: 1  f1_score: 0.6277570410587038
k: 34  k_fold: 2  f1_score: 0.6277570410587038
k: 34  k_fold: 3  f1_score: 0.6277570410587038
k: 34  k_fold: 4  f1_score: 0.6277570410587038
k: 34  k_fold: 5  f1_score: 0.6277570410587038
k: 34  k_fold: 6  f1_score: 0.6277570410587038
k: 34  k_fold: 7  f1_score: 0.6277570410587038
k: 34  k_fold: 8  f1_score: 0.6277570410587038
k: 34  k_fold: 9  f1_score: 0.6277570410587038
k: 34  k_fold: 10  f1_score: 0.6277570410587038
k: 34  Mean Acc: 0.6277570410587038
k: 35  k_fold: 1  f1_score: 0.6260604004071938
```

```
k: 35  k_fold: 2  f1_score: 0.6260604004071938
k: 35  k_fold: 3  f1_score: 0.6260604004071938
k: 35  k_fold: 4  f1_score: 0.6260604004071938
k: 35  k_fold: 5  f1_score: 0.6260604004071938
k: 35  k_fold: 6  f1_score: 0.6260604004071938
k: 35  k_fold: 7  f1_score: 0.6260604004071938
k: 35  k_fold: 8  f1_score: 0.6260604004071938
k: 35  k_fold: 9  f1_score: 0.6260604004071938
k: 35  k_fold: 10  f1_score: 0.6260604004071938
k: 35  Mean Acc: 0.6260604004071937
k: 36  k_fold: 1  f1_score: 0.6250424160162877
k: 36  k_fold: 2  f1_score: 0.6250424160162877
k: 36  k_fold: 3  f1_score: 0.6250424160162877
k: 36  k_fold: 4  f1_score: 0.6250424160162877
k: 36  k_fold: 5  f1_score: 0.6250424160162877
k: 36  k_fold: 6  f1_score: 0.6250424160162877
k: 36  k_fold: 7  f1_score: 0.6250424160162877
k: 36  k_fold: 8  f1_score: 0.6250424160162877
k: 36  k_fold: 9  f1_score: 0.6250424160162877
k: 36  k_fold: 10  f1_score: 0.6250424160162877
k: 36  Mean Acc: 0.6250424160162877
k: 37  k_fold: 1  f1_score: 0.6260604004071938
k: 37  k_fold: 2  f1_score: 0.6260604004071938
k: 37  k_fold: 3  f1_score: 0.6260604004071938
k: 37  k_fold: 4  f1_score: 0.6260604004071938
k: 37  k_fold: 5  f1_score: 0.6260604004071938
k: 37  k_fold: 6  f1_score: 0.6260604004071938
k: 37  k_fold: 7  f1_score: 0.6260604004071938
k: 37  k_fold: 8  f1_score: 0.6260604004071938
k: 37  k_fold: 9  f1_score: 0.6260604004071938
k: 37  k_fold: 10  f1_score: 0.6260604004071938
k: 37  Mean Acc: 0.6260604004071937
k: 38  k_fold: 1  f1_score: 0.6243637597556837
k: 38  k_fold: 2  f1_score: 0.6243637597556837
k: 38  k_fold: 3  f1_score: 0.6243637597556837
k: 38  k_fold: 4  f1_score: 0.6243637597556837
k: 38  k_fold: 5  f1_score: 0.6243637597556837
k: 38  k_fold: 6  f1_score: 0.6243637597556837
k: 38  k_fold: 7  f1_score: 0.6243637597556837
k: 38  k_fold: 8  f1_score: 0.6243637597556837
k: 38  k_fold: 9  f1_score: 0.6243637597556837
k: 38  k_fold: 10  f1_score: 0.6243637597556837
k: 38  Mean Acc: 0.6243637597556837
k: 39  k_fold: 1  f1_score: 0.6277570410587038
k: 39  k_fold: 2  f1_score: 0.6277570410587038
k: 39  k_fold: 3  f1_score: 0.6277570410587038
k: 39  k_fold: 4  f1_score: 0.6277570410587038
k: 39  k_fold: 5  f1_score: 0.6277570410587038
k: 39  k_fold: 6  f1_score: 0.6277570410587038
k: 39  k_fold: 7  f1_score: 0.6277570410587038
k: 39  k_fold: 8  f1_score: 0.6277570410587038
k: 39  k_fold: 9  f1_score: 0.6277570410587038
k: 39  k_fold: 10  f1_score: 0.6277570410587038
k: 39  Mean Acc: 0.6277570410587038
k: 40  k_fold: 1  f1_score: 0.6240244316253818
k: 40  k_fold: 2  f1_score: 0.6240244316253818
k: 40  k_fold: 3  f1_score: 0.6240244316253818
```

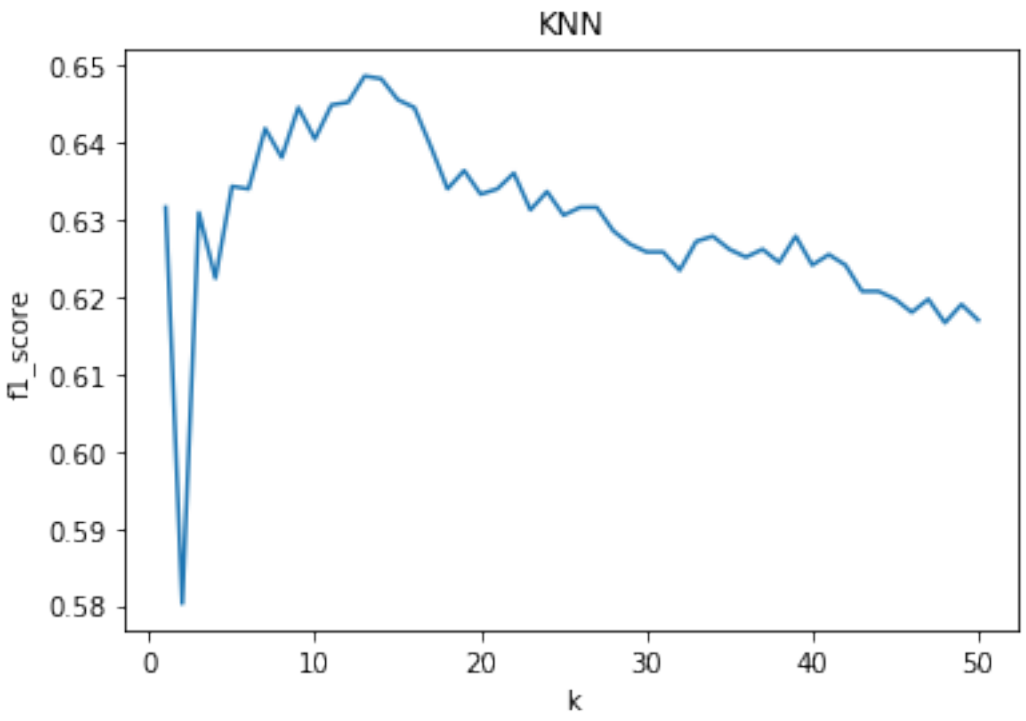
```
k: 40  k_fold: 4  f1_score: 0.6240244316253818
k: 40  k_fold: 5  f1_score: 0.6240244316253818
k: 40  k_fold: 6  f1_score: 0.6240244316253818
k: 40  k_fold: 7  f1_score: 0.6240244316253818
k: 40  k_fold: 8  f1_score: 0.6240244316253818
k: 40  k_fold: 9  f1_score: 0.6240244316253818
k: 40  k_fold: 10  f1_score: 0.6240244316253818
k: 40  Mean Acc: 0.6240244316253818
k: 41  k_fold: 1  f1_score: 0.6253817441465898
k: 41  k_fold: 2  f1_score: 0.6253817441465898
k: 41  k_fold: 3  f1_score: 0.6253817441465898
k: 41  k_fold: 4  f1_score: 0.6253817441465898
k: 41  k_fold: 5  f1_score: 0.6253817441465898
k: 41  k_fold: 6  f1_score: 0.6253817441465898
k: 41  k_fold: 7  f1_score: 0.6253817441465898
k: 41  k_fold: 8  f1_score: 0.6253817441465898
k: 41  k_fold: 9  f1_score: 0.6253817441465898
k: 41  k_fold: 10  f1_score: 0.6253817441465898
k: 41  Mean Acc: 0.6253817441465898
k: 42  k_fold: 1  f1_score: 0.6240244316253818
k: 42  k_fold: 2  f1_score: 0.6240244316253818
k: 42  k_fold: 3  f1_score: 0.6240244316253818
k: 42  k_fold: 4  f1_score: 0.6240244316253818
k: 42  k_fold: 5  f1_score: 0.6240244316253818
k: 42  k_fold: 6  f1_score: 0.6240244316253818
k: 42  k_fold: 7  f1_score: 0.6240244316253818
k: 42  k_fold: 8  f1_score: 0.6240244316253818
k: 42  k_fold: 9  f1_score: 0.6240244316253818
k: 42  k_fold: 10  f1_score: 0.6240244316253818
k: 42  Mean Acc: 0.6240244316253818
k: 43  k_fold: 1  f1_score: 0.6206311503223617
k: 43  k_fold: 2  f1_score: 0.6206311503223617
k: 43  k_fold: 3  f1_score: 0.6206311503223617
k: 43  k_fold: 4  f1_score: 0.6206311503223617
k: 43  k_fold: 5  f1_score: 0.6206311503223617
k: 43  k_fold: 6  f1_score: 0.6206311503223617
k: 43  k_fold: 7  f1_score: 0.6206311503223617
k: 43  k_fold: 8  f1_score: 0.6206311503223617
k: 43  k_fold: 9  f1_score: 0.6206311503223617
k: 43  k_fold: 10  f1_score: 0.6206311503223617
k: 43  Mean Acc: 0.6206311503223618
k: 44  k_fold: 1  f1_score: 0.6206311503223617
k: 44  k_fold: 2  f1_score: 0.6206311503223617
k: 44  k_fold: 3  f1_score: 0.6206311503223617
k: 44  k_fold: 4  f1_score: 0.6206311503223617
k: 44  k_fold: 5  f1_score: 0.6206311503223617
k: 44  k_fold: 6  f1_score: 0.6206311503223617
k: 44  k_fold: 7  f1_score: 0.6206311503223617
k: 44  k_fold: 8  f1_score: 0.6206311503223617
k: 44  k_fold: 9  f1_score: 0.6206311503223617
k: 44  k_fold: 10  f1_score: 0.6206311503223617
k: 44  Mean Acc: 0.6206311503223618
k: 45  k_fold: 1  f1_score: 0.6196131659314558
k: 45  k_fold: 2  f1_score: 0.6196131659314558
k: 45  k_fold: 3  f1_score: 0.6196131659314558
k: 45  k_fold: 4  f1_score: 0.6196131659314558
k: 45  k_fold: 5  f1_score: 0.6196131659314558
```

```
k: 45  k_fold: 6  f1_score: 0.6196131659314558
k: 45  k_fold: 7  f1_score: 0.6196131659314558
k: 45  k_fold: 8  f1_score: 0.6196131659314558
k: 45  k_fold: 9  f1_score: 0.6196131659314558
k: 45  k_fold: 10  f1_score: 0.6196131659314558
k: 45  Mean Acc: 0.6196131659314557
k: 46  k_fold: 1  f1_score: 0.6179165252799457
k: 46  k_fold: 2  f1_score: 0.6179165252799457
k: 46  k_fold: 3  f1_score: 0.6179165252799457
k: 46  k_fold: 4  f1_score: 0.6179165252799457
k: 46  k_fold: 5  f1_score: 0.6179165252799457
k: 46  k_fold: 6  f1_score: 0.6179165252799457
k: 46  k_fold: 7  f1_score: 0.6179165252799457
k: 46  k_fold: 8  f1_score: 0.6179165252799457
k: 46  k_fold: 9  f1_score: 0.6179165252799457
k: 46  k_fold: 10  f1_score: 0.6179165252799457
k: 46  Mean Acc: 0.6179165252799457
k: 47  k_fold: 1  f1_score: 0.6196131659314558
k: 47  k_fold: 2  f1_score: 0.6196131659314558
k: 47  k_fold: 3  f1_score: 0.6196131659314558
k: 47  k_fold: 4  f1_score: 0.6196131659314558
k: 47  k_fold: 5  f1_score: 0.6196131659314558
k: 47  k_fold: 6  f1_score: 0.6196131659314558
k: 47  k_fold: 7  f1_score: 0.6196131659314558
k: 47  k_fold: 8  f1_score: 0.6196131659314558
k: 47  k_fold: 9  f1_score: 0.6196131659314558
k: 47  k_fold: 10  f1_score: 0.6196131659314558
k: 47  Mean Acc: 0.6196131659314557
k: 48  k_fold: 1  f1_score: 0.6165592127587377
k: 48  k_fold: 2  f1_score: 0.6165592127587377
k: 48  k_fold: 3  f1_score: 0.6165592127587377
k: 48  k_fold: 4  f1_score: 0.6165592127587377
k: 48  k_fold: 5  f1_score: 0.6165592127587377
k: 48  k_fold: 6  f1_score: 0.6165592127587377
k: 48  k_fold: 7  f1_score: 0.6165592127587377
k: 48  k_fold: 8  f1_score: 0.6165592127587377
k: 48  k_fold: 9  f1_score: 0.6165592127587377
k: 48  k_fold: 10  f1_score: 0.6165592127587377
k: 48  Mean Acc: 0.6165592127587376
k: 49  k_fold: 1  f1_score: 0.6189345096708517
k: 49  k_fold: 2  f1_score: 0.6189345096708517
k: 49  k_fold: 3  f1_score: 0.6189345096708517
k: 49  k_fold: 4  f1_score: 0.6189345096708517
k: 49  k_fold: 5  f1_score: 0.6189345096708517
k: 49  k_fold: 6  f1_score: 0.6189345096708517
k: 49  k_fold: 7  f1_score: 0.6189345096708517
k: 49  k_fold: 8  f1_score: 0.6189345096708517
k: 49  k_fold: 9  f1_score: 0.6189345096708517
k: 49  k_fold: 10  f1_score: 0.6189345096708517
k: 49  Mean Acc: 0.6189345096708517
k: 50  k_fold: 1  f1_score: 0.6168985408890397
k: 50  k_fold: 2  f1_score: 0.6168985408890397
k: 50  k_fold: 3  f1_score: 0.6168985408890397
k: 50  k_fold: 4  f1_score: 0.6168985408890397
k: 50  k_fold: 5  f1_score: 0.6168985408890397
k: 50  k_fold: 6  f1_score: 0.6168985408890397
k: 50  k_fold: 7  f1_score: 0.6168985408890397
```

```
k: 50   k_fold: 8   f1_score: 0.6168985408890397
k: 50   k_fold: 9   f1_score: 0.6168985408890397
k: 50   k_fold: 10   f1_score: 0.6168985408890397
k: 50   Mean Acc: 0.6168985408890396
```

In [38]:

```
import matplotlib.pyplot as plt
plt.plot(np.arange(1,len(acc)+1),acc)
plt.ylabel('f1_score')
plt.xlabel('k')
plt.title("KNN")
plt.show()
```



In [39]:

```
bestk = np.argmax(acc)+1
print('Best k:',bestk)


knn_Best2 = KNeighborsClassifier(n_neighbors=bestk)
knn_Best2.fit(X_Train,np.array(Y_Train))
pred_Best2 = knn_Best2.predict(X_Test)
f1_Best2 = f1_score(np.array(Y_Test),pred_Best2,average='micro')
print ('F1 Score for Best k scenario:',f1_Best2)
accu_Best2 = accuracy_score(np.array(Y_Test),pred_Best2)
print ('Accuracy for Best k scenario:',accu_Best2)
C_M_Best2 = confusion_matrix(np.array(Y_Test),pred_Best2)
print ('Confusion Matrix for Best k scenario:\n',C_M_Best2)
```

```
Best k: 13
F1 Score for Best k scenario: 0.6484560570071259
Accuracy for Best k scenario: 0.6484560570071259
Confusion Matrix for Best k scenario:
 [[452  20  24   0   0   0]
 [232 200  38   0   1   0]
 [172  40 208   0   0   0]
 [  0   4   1 289 170  27]
 [  5   6   1 103 389  28]
 [  5   1   1  45 112 373]]
```

In [41]:

```
# Part 3: Multiclass Logistic Regression with Elastic Net #
# Build an elastic-net regularized logistic regression classifier for this dat
a. #

from sklearn.linear_model import SGDClassifier
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

alphas = [1e-4,3e-4,1e-3,3e-3,1e-2,3e-2]
l1_ratios = [0,0.15,0.5,0.7,1]

accP3 = []
a_l1 = []
for alpha in alphas:
    for l1 in l1_ratios:
        #lre = ElasticNet(alpha=alpha,l1_ratio=l1)
        enet = SGDClassifier(alpha=alpha,l1_ratio=l1,penalty="elasticnet")
        scoresP3 = []
        for i in np.arange(1,11):
            #Xtrain, Xtest, ytrain, ytest = train_test_split(X_train, y_train)
            #enet.fit(X_Train,np.array(Y_Train))
            enet.fit(X_Train,Y_Train.values.reshape(Y_Train.shape[0],))
            predP3 = enet.predict(X_Test)
            print (predP3)
            f1_enet = f1_score(np.array(Y_Test),np.array(predP3),average='micr
o')
            print ('alpha:',alpha,'l1:',l1,' k_fold:',i,' f1_score:',f1_enet)
            scoresP3.append(f1_enet)
        maccP3 = np.mean(scoresP3)
        print ( 'Mean F1_enet:',maccP3)
        a_l1.append((alpha,l1))
        accP3.append(maccP3)
```

```
['5' '4' '5' ... '2' '3' '1']
alpha: 0.0001 l1: 0  k_fold: 1  f1_score: 0.7495758398371225
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0  k_fold: 2  f1_score: 0.7750254496097727
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0  k_fold: 3  f1_score: 0.7678995588734306
['6' '4' '4' ... '2' '2' '1']
alpha: 0.0001 l1: 0  k_fold: 4  f1_score: 0.6810315575161181
['5' '4' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 0  k_fold: 5  f1_score: 0.7645062775704106
['5' '5' '5' ... '2' '2' '3']
alpha: 0.0001 l1: 0  k_fold: 6  f1_score: 0.7458432304038005
['6' '5' '5' ... '2' '2' '3']
alpha: 0.0001 l1: 0  k_fold: 7  f1_score: 0.7465218866644044
['6' '4' '6' ... '2' '2' '2']
alpha: 0.0001 l1: 0  k_fold: 8  f1_score: 0.7767220902612827
['5' '5' '5' ... '2' '2' '3']
alpha: 0.0001 l1: 0  k_fold: 9  f1_score: 0.7621309806582965
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0  k_fold: 10  f1_score: 0.7424499491007804
Mean F1_enet: 0.751170682049542
```

```
['5' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.15  k_fold: 1  f1_score: 0.7125890736342043
['6' '4' '6' ... '2' '2' '1']
alpha: 0.0001 l1: 0.15  k_fold: 2  f1_score: 0.7105531048523923
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.15  k_fold: 3  f1_score: 0.7594163556158806
['6' '4' '6' ... '2' '3' '1']
alpha: 0.0001 l1: 0.15  k_fold: 4  f1_score: 0.7343060739735325
['5' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.15  k_fold: 5  f1_score: 0.7410926365795725
['6' '5' '6' ... '2' '2' '1']
alpha: 0.0001 l1: 0.15  k_fold: 6  f1_score: 0.7176789955887344
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 0.15  k_fold: 7  f1_score: 0.7689175432643366
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 0.15  k_fold: 8  f1_score: 0.7482185273159146
['5' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.15  k_fold: 9  f1_score: 0.7441465897522904
['6' '5' '6' ... '2' '2' '3']
alpha: 0.0001 l1: 0.15  k_fold: 10  f1_score: 0.7329487614523243
Mean F1_enet: 0.7369867662029183
['6' '5' '5' ... '2' '3' '1']
alpha: 0.0001 l1: 0.5  k_fold: 1  f1_score: 0.7360027146250424
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 2  f1_score: 0.7668815744825246
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 3  f1_score: 0.7784187309127927
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 0.5  k_fold: 4  f1_score: 0.7695961995249406
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 5  f1_score: 0.7719714964370546
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 6  f1_score: 0.7902952154733628
['6' '4' '4' ... '2' '2' '1']
alpha: 0.0001 l1: 0.5  k_fold: 7  f1_score: 0.7387173396674585
['5' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 8  f1_score: 0.7947064811672888
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.5  k_fold: 9  f1_score: 0.8028503562945368
['6' '5' '5' ... '2' '3' '1']
alpha: 0.0001 l1: 0.5  k_fold: 10  f1_score: 0.7869019341703427
Mean F1_enet: 0.7736342042755344
['6' '5' '5' ... '2' '3' '3']
alpha: 0.0001 l1: 0.7  k_fold: 1  f1_score: 0.7729894808279605
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 2  f1_score: 0.7909738717339667
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 3  f1_score: 0.7865626060400406
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 4  f1_score: 0.7804546996946047
['6' '5' '5' ... '2' '2' '3']
alpha: 0.0001 l1: 0.7  k_fold: 5  f1_score: 0.7885985748218527
['6' '4' '6' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 6  f1_score: 0.7787580590430947
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 7  f1_score: 0.7875805904309467
['6' '4' '5' ... '2' '2' '2']
```

```
alpha: 0.0001 l1: 0.7  k_fold: 8  f1_score: 0.7899558873430608
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 0.7  k_fold: 9  f1_score: 0.7936884967763828
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 0.7  k_fold: 10  f1_score: 0.7936884967763828
Mean F1_enet: 0.7863250763488294
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 1  k_fold: 1  f1_score: 0.7801153715643027
['6' '5' '5' ... '2' '2' '3']
alpha: 0.0001 l1: 1  k_fold: 2  f1_score: 0.7940278249066848
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 3  f1_score: 0.7750254496097727
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 4  f1_score: 0.7797760434340006
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 5  f1_score: 0.7753647777400746
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 6  f1_score: 0.7726501526976587
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 7  f1_score: 0.7906345436036647
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 8  f1_score: 0.7835086528673227
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0001 l1: 1  k_fold: 9  f1_score: 0.7740074652188665
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0001 l1: 1  k_fold: 10  f1_score: 0.7940278249066848
Mean F1_enet: 0.7819138106549033
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 1  f1_score: 0.7763827621309807
['6' '4' '4' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 2  f1_score: 0.7312521208008144
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 3  f1_score: 0.7587376993552765
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 4  f1_score: 0.7763827621309807
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 5  f1_score: 0.7770614183915847
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0  k_fold: 6  f1_score: 0.7658635900916186
['6' '5' '5' ... '2' '2' '3']
alpha: 0.0003 l1: 0  k_fold: 7  f1_score: 0.7665422463522227
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0  k_fold: 8  f1_score: 0.7746861214794707
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 9  f1_score: 0.7753647777400746
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0  k_fold: 10  f1_score: 0.7804546996946047
Mean F1_enet: 0.7682728198167629
['5' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.15  k_fold: 1  f1_score: 0.7641669494401085
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.15  k_fold: 2  f1_score: 0.7841873091279267
['6' '4' '6' ... '2' '2' '1']
alpha: 0.0003 l1: 0.15  k_fold: 3  f1_score: 0.7119104173736003
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.15  k_fold: 4  f1_score: 0.7821513403461147
['6' '5' '5' ... '2' '2' '3']
```

```
alpha: 0.0003 l1: 0.15  k_fold: 5  f1_score: 0.7607736681370886
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.15  k_fold: 6  f1_score: 0.7794367153036986
['5' '4' '4' ... '2' '2' '2']
alpha: 0.0003 l1: 0.15  k_fold: 7  f1_score: 0.7634882931795046
['6' '5' '6' ... '2' '2' '2']
alpha: 0.0003 l1: 0.15  k_fold: 8  f1_score: 0.7441465897522904
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.15  k_fold: 9  f1_score: 0.7621309806582965
['6' '4' '5' ... '2' '2' '3']
alpha: 0.0003 l1: 0.15  k_fold: 10  f1_score: 0.7611129962673906
Mean F1_enet: 0.761350525958602
['5' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 1  f1_score: 0.7706141839158466
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.5  k_fold: 2  f1_score: 0.7780794027824907
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 3  f1_score: 0.7919918561248728
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 4  f1_score: 0.7841873091279267
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 5  f1_score: 0.7797760434340006
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 6  f1_score: 0.7828299966067187
['6' '5' '5' ... '2' '3' '2']
alpha: 0.0003 l1: 0.5  k_fold: 7  f1_score: 0.7777400746521886
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.5  k_fold: 8  f1_score: 0.7743467933491687
['6' '4' '4' ... '2' '2' '2']
alpha: 0.0003 l1: 0.5  k_fold: 9  f1_score: 0.7285374957583982
['6' '4' '6' ... '2' '2' '1']
alpha: 0.0003 l1: 0.5  k_fold: 10  f1_score: 0.7719714964370546
Mean F1_enet: 0.7740074652188667
['6' '5' '6' ... '2' '3' '1']
alpha: 0.0003 l1: 0.7  k_fold: 1  f1_score: 0.7794367153036986
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 2  f1_score: 0.7767220902612827
['6' '4' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 0.7  k_fold: 3  f1_score: 0.7882592466915506
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 4  f1_score: 0.7940278249066848
['6' '4' '5' ... '2' '3' '1']
alpha: 0.0003 l1: 0.7  k_fold: 5  f1_score: 0.7709535120461486
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 6  f1_score: 0.7797760434340006
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 7  f1_score: 0.7824906684764167
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 8  f1_score: 0.7702748557855447
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 9  f1_score: 0.7902952154733628
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 0.7  k_fold: 10  f1_score: 0.7719714964370546
Mean F1_enet: 0.7804207668815745
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 1  k_fold: 1  f1_score: 0.7712928401764506
['6' '5' '5' ... '2' '2' '1']
```

```
alpha: 0.0003 l1: 1  k_fold: 2  f1_score: 0.7882592466915506
['6' '5' '5' ... '2' '2' '3']
alpha: 0.0003 l1: 1  k_fold: 3  f1_score: 0.7824906684764167
['6' '5' '5' ... '2' '2' '2']
alpha: 0.0003 l1: 1  k_fold: 4  f1_score: 0.7726501526976587
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 5  f1_score: 0.7872412623006447
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 6  f1_score: 0.7716321683067526
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 7  f1_score: 0.7801153715643027
['6' '4' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 8  f1_score: 0.7790973871733967
['6' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 9  f1_score: 0.7926705123854769
['4' '5' '5' ... '2' '2' '1']
alpha: 0.0003 l1: 1  k_fold: 10  f1_score: 0.7740074652188665
Mean F1_enet: 0.7799457074991516
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 1  f1_score: 0.7848659653885307
['6' '4' '6' ... '2' '2' '1']
alpha: 0.001 l1: 0  k_fold: 2  f1_score: 0.7461825585341024
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 3  f1_score: 0.7828299966067187
['6' '4' '4' ... '2' '2' '1']
alpha: 0.001 l1: 0  k_fold: 4  f1_score: 0.6834068544282321
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 5  f1_score: 0.7590770274855786
['6' '4' '6' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 6  f1_score: 0.7617916525279945
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 7  f1_score: 0.7947064811672888
['5' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0  k_fold: 8  f1_score: 0.7563624024431627
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0  k_fold: 9  f1_score: 0.7743467933491687
['5' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0  k_fold: 10  f1_score: 0.7485578554462164
Mean F1_enet: 0.7592127587376994
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 1  f1_score: 0.7872412623006447
['6' '5' '6' ... '2' '2' '1']
alpha: 0.001 l1: 0.15  k_fold: 2  f1_score: 0.7628096369189006
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 3  f1_score: 0.7784187309127927
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 4  f1_score: 0.7818120122158126
['6' '4' '4' ... '2' '2' '1']
alpha: 0.001 l1: 0.15  k_fold: 5  f1_score: 0.7349847302341365
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 6  f1_score: 0.7699355276552428
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 7  f1_score: 0.7760434340006787
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.15  k_fold: 8  f1_score: 0.7767220902612827
['6' '4' '6' ... '2' '2' '1']
alpha: 0.001 l1: 0.15  k_fold: 9  f1_score: 0.7594163556158806
```

```
['6' '4' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.15  k_fold: 10  f1_score: 0.7590770274855786
Mean F1_enet: 0.768646080760095
['5' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 1  f1_score: 0.7651849338310145
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.5  k_fold: 2  f1_score: 0.7695961995249406
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 3  f1_score: 0.7767220902612827
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 4  f1_score: 0.7594163556158806
['5' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 5  f1_score: 0.7363420427553444
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.5  k_fold: 6  f1_score: 0.7533084492704445
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.5  k_fold: 7  f1_score: 0.7896165592127586
['6' '4' '4' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 8  f1_score: 0.7678995588734306
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.5  k_fold: 9  f1_score: 0.7522904648795385
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.5  k_fold: 10  f1_score: 0.7926705123854769
Mean F1_enet: 0.7663047166610113
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 1  f1_score: 0.7780794027824907
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.7  k_fold: 2  f1_score: 0.7665422463522227
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 0.7  k_fold: 3  f1_score: 0.7658635900916186
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 4  f1_score: 0.7760434340006787
['6' '5' '6' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 5  f1_score: 0.7590770274855786
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 6  f1_score: 0.7746861214794707
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 7  f1_score: 0.7852052935188328
['6' '4' '4' ... '2' '2' '1']
alpha: 0.001 l1: 0.7  k_fold: 8  f1_score: 0.7125890736342043
['6' '4' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 9  f1_score: 0.7909738717339667
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 0.7  k_fold: 10  f1_score: 0.7699355276552428
Mean F1_enet: 0.7678995588734306
['6' '4' '4' ... '2' '3' '1']
alpha: 0.001 l1: 1  k_fold: 1  f1_score: 0.7550050899219545
['6' '4' '4' ... '2' '2' '2']
alpha: 0.001 l1: 1  k_fold: 2  f1_score: 0.7583983712249746
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 3  f1_score: 0.7678995588734306
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 4  f1_score: 0.7699355276552428
['6' '4' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 5  f1_score: 0.7536477774007465
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 1  k_fold: 6  f1_score: 0.7628096369189006
```

```
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 7  f1_score: 0.7607736681370886
['6' '5' '5' ... '2' '2' '2']
alpha: 0.001 l1: 1  k_fold: 8  f1_score: 0.7526297930098406
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 9  f1_score: 0.7628096369189006
['6' '5' '5' ... '2' '2' '1']
alpha: 0.001 l1: 1  k_fold: 10  f1_score: 0.7621309806582965
Mean F1_enet: 0.7606040040719375
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 1  f1_score: 0.7753647777400746
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 2  f1_score: 0.7719714964370546
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 3  f1_score: 0.7662029182219205
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 4  f1_score: 0.7896165592127586
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0  k_fold: 5  f1_score: 0.7706141839158466
['6' '4' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0  k_fold: 6  f1_score: 0.7438072616219885
['6' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 7  f1_score: 0.7950458092975908
['6' '5' '6' ... '2' '2' '2']
alpha: 0.003 l1: 0  k_fold: 8  f1_score: 0.7281981676280964
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0  k_fold: 9  f1_score: 0.7594163556158806
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0  k_fold: 10  f1_score: 0.7675602307431286
Mean F1_enet: 0.766779776043434
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.15  k_fold: 1  f1_score: 0.7872412623006447
['6' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.15  k_fold: 2  f1_score: 0.7634882931795046
['6' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.15  k_fold: 3  f1_score: 0.7706141839158466
['6' '4' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.15  k_fold: 4  f1_score: 0.7668815744825246
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.15  k_fold: 5  f1_score: 0.7655242619613166
['6' '4' '4' ... '2' '2' '1']
alpha: 0.003 l1: 0.15  k_fold: 6  f1_score: 0.7468612147947065
['6' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.15  k_fold: 7  f1_score: 0.7760434340006787
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.15  k_fold: 8  f1_score: 0.7594163556158806
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.15  k_fold: 9  f1_score: 0.7638276213098066
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.15  k_fold: 10  f1_score: 0.7665422463522227
Mean F1_enet: 0.7666440447913131
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.5  k_fold: 1  f1_score: 0.7604343400067867
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.5  k_fold: 2  f1_score: 0.7434679334916865
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.5  k_fold: 3  f1_score: 0.7519511367492366
```

['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.5 k_fold: 4 f1_score: 0.7512724804886325
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.5 k_fold: 5 f1_score: 0.7227689175432643
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.5 k_fold: 6 f1_score: 0.7553444180522564
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.5 k_fold: 7 f1_score: 0.7594163556158806
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.5 k_fold: 8 f1_score: 0.7604343400067867
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.5 k_fold: 9 f1_score: 0.7237869019341704
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.5 k_fold: 10 f1_score: 0.7373600271462503
Mean F1_enet: 0.7466236851034951
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 0.7 k_fold: 1 f1_score: 0.7298948082796064
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 2 f1_score: 0.7451645741431965
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 3 f1_score: 0.7421106209704784
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 4 f1_score: 0.7329487614523243
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 5 f1_score: 0.7546657617916527
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 6 f1_score: 0.7519511367492366
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 7 f1_score: 0.7475398710553105
['5' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 8 f1_score: 0.7407533084492705
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 9 f1_score: 0.7397353240583644
['5' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 0.7 k_fold: 10 f1_score: 0.7448252460128945
Mean F1_enet: 0.7429589412962334
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 1 f1_score: 0.7034272141160502
['5' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 1 k_fold: 2 f1_score: 0.7159823549372243
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 3 f1_score: 0.7207329487614523
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 4 f1_score: 0.7081778079402782
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 5 f1_score: 0.7200542925008484
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 6 f1_score: 0.7261621988462843
['6' '5' '5' ... '2' '2' '1']
alpha: 0.003 l1: 1 k_fold: 7 f1_score: 0.7054631828978623
['6' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 8 f1_score: 0.7326094333220224
['6' '4' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 9 f1_score: 0.7251442144553784
['1' '5' '5' ... '2' '2' '2']
alpha: 0.003 l1: 1 k_fold: 10 f1_score: 0.7217509331523583
Mean F1_enet: 0.7179504580929759

```
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 1  f1_score: 0.7465218866644044
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 2  f1_score: 0.7441465897522904
['6' '4' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0  k_fold: 3  f1_score: 0.7522904648795385
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0  k_fold: 4  f1_score: 0.7502544960977267
['6' '4' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 5  f1_score: 0.7516118086189345
['6' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 6  f1_score: 0.7363420427553444
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 7  f1_score: 0.7431286053613844
['5' '4' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0  k_fold: 8  f1_score: 0.7400746521886664
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 9  f1_score: 0.7332880895826265
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0  k_fold: 10  f1_score: 0.7393959959280624
Mean F1_enet: 0.7437054631828979
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 1  f1_score: 0.7356633864947404
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 2  f1_score: 0.7217509331523583
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 3  f1_score: 0.7261621988462843
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 4  f1_score: 0.7227689175432643
['5' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0.15  k_fold: 5  f1_score: 0.7142857142857143
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 6  f1_score: 0.7237869019341704
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 7  f1_score: 0.7068204954190702
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 8  f1_score: 0.7360027146250424
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 9  f1_score: 0.7285374957583982
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.15  k_fold: 10  f1_score: 0.7248048863250763
Mean F1_enet: 0.724058364438412
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 1  f1_score: 0.665761791652528
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 2  f1_score: 0.66949440108585
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 3  f1_score: 0.668137088564642
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 4  f1_score: 0.666779776043434
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 5  f1_score: 0.6786562606040041
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 6  f1_score: 0.654903291482864
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 7  f1_score: 0.6582965727858839
['5' '5' '5' ... '2' '2' '2']
```

```
alpha: 0.01 l1: 0.5  k_fold: 8  f1_score: 0.666440447913132
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 9  f1_score: 0.6623685103495079
['5' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 0.5  k_fold: 10  f1_score: 0.660671869697998
Mean F1_enet: 0.6651510010179844
['6' '5' '5' ... '2' '2' '3']
alpha: 0.01 l1: 0.7  k_fold: 1  f1_score: 0.6179165252799457
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 0.7  k_fold: 2  f1_score: 0.6138445877163217
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 0.7  k_fold: 3  f1_score: 0.6124872751951137
['6' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0.7  k_fold: 4  f1_score: 0.6321683067526298
['6' '5' '5' ... '1' '1' '1']
alpha: 0.01 l1: 0.7  k_fold: 5  f1_score: 0.5978961655921275
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 0.7  k_fold: 6  f1_score: 0.6084153376314897
['6' '5' '5' ... '2' '1' '3']
alpha: 0.01 l1: 0.7  k_fold: 7  f1_score: 0.6094333220223956
['6' '5' '5' ... '2' '2' '1']
alpha: 0.01 l1: 0.7  k_fold: 8  f1_score: 0.6131659314557176
['6' '5' '5' ... '2' '1' '3']
alpha: 0.01 l1: 0.7  k_fold: 9  f1_score: 0.6107906345436037
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 0.7  k_fold: 10  f1_score: 0.6060400407193757
Mean F1_enet: 0.6122158126908721
['6' '5' '5' ... '2' '2' '3']
alpha: 0.01 l1: 1  k_fold: 1  f1_score: 0.5850016966406515
['6' '4' '4' ... '1' '1' '3']
alpha: 0.01 l1: 1  k_fold: 2  f1_score: 0.5755005089921954
['5' '5' '5' ... '2' '2' '3']
alpha: 0.01 l1: 1  k_fold: 3  f1_score: 0.5934848998982015
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 1  k_fold: 4  f1_score: 0.5833050559891415
['5' '5' '5' ... '1' '1' '1']
alpha: 0.01 l1: 1  k_fold: 5  f1_score: 0.5697319307770614
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 1  k_fold: 6  f1_score: 0.5928062436375976
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 1  k_fold: 7  f1_score: 0.5975568374618256
['5' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 1  k_fold: 8  f1_score: 0.5836443841194435
['6' '5' '5' ... '2' '1' '1']
alpha: 0.01 l1: 1  k_fold: 9  f1_score: 0.5978961655921275
['6' '5' '5' ... '2' '2' '2']
alpha: 0.01 l1: 1  k_fold: 10  f1_score: 0.6070580251102816
Mean F1_enet: 0.5885985748218526
['5' '5' '5' ... '2' '2' '1']
alpha: 0.03 l1: 0  k_fold: 1  f1_score: 0.6854428232100441
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 2  f1_score: 0.7030878859857482
['5' '5' '5' ... '2' '2' '1']
alpha: 0.03 l1: 0  k_fold: 3  f1_score: 0.6973193077706142
['6' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 4  f1_score: 0.7068204954190702
['6' '5' '5' ... '2' '2' '1']
```

```
alpha: 0.03 l1: 0  k_fold: 5  f1_score: 0.6996946046827281
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 6  f1_score: 0.6996946046827281
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 7  f1_score: 0.6963013233797082
['6' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 8  f1_score: 0.7007125890736342
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0  k_fold: 9  f1_score: 0.6915507295554801
['5' '5' '5' ... '2' '2' '1']
alpha: 0.03 l1: 0  k_fold: 10  f1_score: 0.6783169324737021
Mean F1_enet: 0.6958941296233458
['5' '5' '5' ... '1' '2' '2']
alpha: 0.03 l1: 0.15  k_fold: 1  f1_score: 0.6165592127587377
['6' '5' '5' ... '1' '1' '2']
alpha: 0.03 l1: 0.15  k_fold: 2  f1_score: 0.6128266033254157
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0.15  k_fold: 3  f1_score: 0.6257210722768918
['5' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0.15  k_fold: 4  f1_score: 0.6253817441465898
['6' '5' '5' ... '1' '1' '1']
alpha: 0.03 l1: 0.15  k_fold: 5  f1_score: 0.6202918221920597
['6' '5' '5' ... '2' '2' '2']
alpha: 0.03 l1: 0.15  k_fold: 6  f1_score: 0.6338649474041398
['6' '5' '5' ... '1' '2' '1']
alpha: 0.03 l1: 0.15  k_fold: 7  f1_score: 0.6128266033254157
['5' '5' '5' ... '2' '2' '1']
alpha: 0.03 l1: 0.15  k_fold: 8  f1_score: 0.6131659314557176
['6' '5' '5' ... '1' '2' '2']
alpha: 0.03 l1: 0.15  k_fold: 9  f1_score: 0.6158805564981337
['5' '5' '5' ... '2' '1' '2']
alpha: 0.03 l1: 0.15  k_fold: 10  f1_score: 0.6158805564981337
Mean F1_enet: 0.6192399049881235
['6' '5' '5' ... '1' '1' '3']
alpha: 0.03 l1: 0.5  k_fold: 1  f1_score: 0.5022056328469631
['6' '5' '5' ... '6' '2' '3']
alpha: 0.03 l1: 0.5  k_fold: 2  f1_score: 0.509670851713607
['6' '5' '5' ... '6' '2' '3']
alpha: 0.03 l1: 0.5  k_fold: 3  f1_score: 0.511706820495419
['6' '5' '5' ... '6' '2' '3']
alpha: 0.03 l1: 0.5  k_fold: 4  f1_score: 0.5191720393620631
['6' '5' '5' ... '2' '1' '3']
alpha: 0.03 l1: 0.5  k_fold: 5  f1_score: 0.5300305395317272
['6' '5' '5' ... '2' '1' '3']
alpha: 0.03 l1: 0.5  k_fold: 6  f1_score: 0.5022056328469631
['6' '5' '5' ... '6' '6' '3']
alpha: 0.03 l1: 0.5  k_fold: 7  f1_score: 0.511367492365117
['6' '5' '5' ... '2' '6' '3']
alpha: 0.03 l1: 0.5  k_fold: 8  f1_score: 0.5249406175771971
['6' '5' '5' ... '2' '2' '3']
alpha: 0.03 l1: 0.5  k_fold: 9  f1_score: 0.5320665083135392
['6' '5' '5' ... '2' '1' '3']
alpha: 0.03 l1: 0.5  k_fold: 10  f1_score: 0.511706820495419
Mean F1_enet: 0.5155072955548015
['6' '6' '6' ... '6' '1' '3']
alpha: 0.03 l1: 0.7  k_fold: 1  f1_score: 0.4648795385137428
['6' '5' '5' ... '6' '1' '3']
```

```
alpha: 0.03 l1: 0.7  k_fold: 2  f1_score: 0.506956226671191
['6' '6' '6' ... '6' '1' '3']
alpha: 0.03 l1: 0.7  k_fold: 3  f1_score: 0.5066168985408891
['6' '5' '5' ... '6' '2' '3']
alpha: 0.03 l1: 0.7  k_fold: 4  f1_score: 0.5184933831014591
['6' '6' '6' ... '6' '2' '3']
alpha: 0.03 l1: 0.7  k_fold: 5  f1_score: 0.47336274177129284
['6' '6' '6' ... '6' '2' '3']
alpha: 0.03 l1: 0.7  k_fold: 6  f1_score: 0.4862572107227689
['6' '5' '5' ... '6' '6' '3']
alpha: 0.03 l1: 0.7  k_fold: 7  f1_score: 0.5127248048863251
['6' '6' '5' ... '6' '1' '3']
alpha: 0.03 l1: 0.7  k_fold: 8  f1_score: 0.46929080420766883
['6' '6' '6' ... '6' '6' '3']
alpha: 0.03 l1: 0.7  k_fold: 9  f1_score: 0.4696301323379708
['6' '6' '6' ... '6' '1' '3']
alpha: 0.03 l1: 0.7  k_fold: 10  f1_score: 0.48829317950458095
Mean F1_enet: 0.489650492025789
['6' '5' '5' ... '2' '6' '2']
alpha: 0.03 l1: 1  k_fold: 1  f1_score: 0.4156769596199525
['5' '5' '5' ... '2' '5' '5']
alpha: 0.03 l1: 1  k_fold: 2  f1_score: 0.34272141160502206
['5' '5' '5' ... '2' '5' '5']
alpha: 0.03 l1: 1  k_fold: 3  f1_score: 0.41839158466236853
['4' '4' '4' ... '4' '4' '4']
alpha: 0.03 l1: 1  k_fold: 4  f1_score: 0.27519511367492366
['4' '4' '4' ... '4' '4' '4']
alpha: 0.03 l1: 1  k_fold: 5  f1_score: 0.27417712928401766
['6' '4' '4' ... '2' '2' '2']
alpha: 0.03 l1: 1  k_fold: 6  f1_score: 0.5042416016287751
['5' '6' '4' ... '1' '2' '1']
alpha: 0.03 l1: 1  k_fold: 7  f1_score: 0.4604682728198168
['5' '5' '5' ... '5' '5' '5']
alpha: 0.03 l1: 1  k_fold: 8  f1_score: 0.39633525619273846
['5' '2' '2' ... '3' '3' '5']
alpha: 0.03 l1: 1  k_fold: 9  f1_score: 0.42992874109263657
['5' '5' '5' ... '1' '1' '1']
alpha: 0.03 l1: 1  k_fold: 10  f1_score: 0.3498473023413641
Mean F1_enet: 0.3866983372921615
```

In [42]:

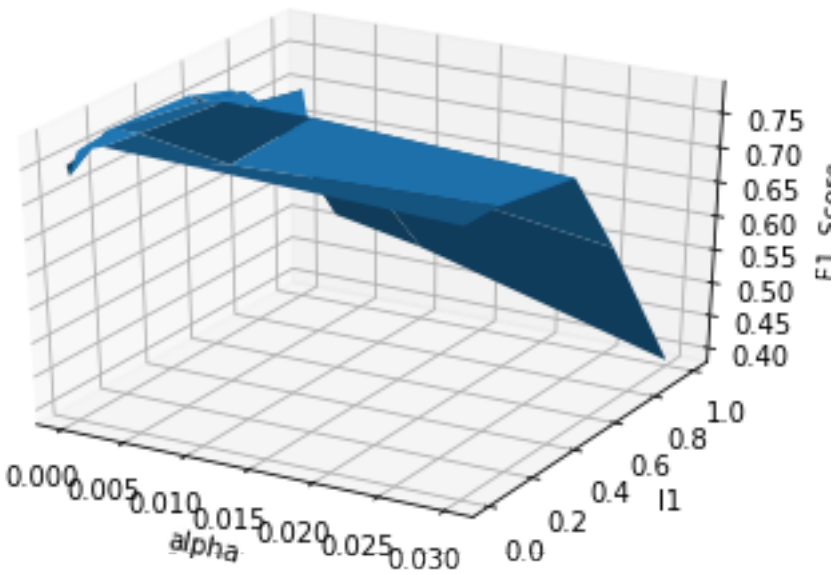
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm

fig_3 = plt.figure()
ax = fig_3.add_subplot(111, projection='3d')
X3, Y3 = np.meshgrid(alphas, l1_ratios)
zs = np.array(accP3)#np.array([fun(x,y) for x,y in zip(np.ravel(X), np.ravel(Y
))])
Z3 = zs.reshape(X3.shape)

ax.plot_surface(X3, Y3, Z3)

ax.set_xlabel('alpha')
ax.set_ylabel('l1')
ax.set_zlabel('F1_Score')

plt.show()
```



In [43]:

```
best_al_l1 = a_l1[np.argmax(accP3)]
print('Best Alpha:',best_al_l1[0], ' Best l1:',best_al_l1[1])
enet_Best3 = SGDClassifier(alpha=best_al_l1[0],l1_ratio=best_al_l1[1],penalty=
"elasticnet")
enet_Best3.fit(X_Train,np.array(Y_Train))
pred_enet_Best3 = enet_Best3.predict(X_Test)
f1_enet_Best3 = f1_score(np.array(Y_Test),pred_enet_Best3,average='micro')
print ('F1 Accuracy for Best Alpha & l1-Ratio:',f1_enet_Best3)
accu_enet_Best3 = accuracy_score(np.array(Y_Test),pred_enet_Best3)
print ('Accuracy for Best Alpha & l1-Ratio:',accu_enet_Best3)
cm_enet_Best3 = confusion_matrix(np.array(Y_Test),pred_enet_Best3)
print ('Confusion Matrix for Best Alpha & l1-Ratio:\n',cm_enet_Best3)
```

```
Best Alpha: 0.0001  Best l1: 0.7
F1 Accuracy for Best Alpha & l1-Ratio: 0.7828299966067187
Accuracy for Best Alpha & l1-Ratio: 0.7828299966067187
Confusion Matrix for Best Alpha & l1-Ratio:
 [[444  18  34   0   0   0]
 [ 76 361  33   0   1   0]
 [ 49  41 330   0   0   0]
 [  0   3   3 261 179  45]
 [  2   0   2  63 395  70]
 [  0   1   0   9  11 516]]
```

In [ ]:

```
# Part 4: Support Vector Machine (RBF Kernel) #
# Build a SVM (with RBF Kernel) classifier for this data. #
```

In [44]:

```
fID = 218187754%3

fID
```

Out[44]:

```
1
```

In [46]:

```
from sklearn import svm
from sklearn.metrics import precision_score

gammas = [1e-3, 1e-4]
CP = [1, 10, 100, 1000]

acc_SVM = []
g_C_SVM = []
for gamma in gammas:
    for C in CP:
        SVM = svm.SVC(kernel='rbf',gamma=gamma,C=C)
        scores_SVM = []
        for i in np.arange(1,11):
            SVM.fit(X_Train,np.array(Y_Train))
            pred_SVM = SVM.predict(X_Test)
            #f1_SVM = f1_score(np.array(Y_Test),pred_SVM,average='micro')
            precision_SVM = precision_score(Y_Test, pred_SVM, average='micro')
            print ('gamma:',gamma,'C:',C,' k_fold:',i,' precision for SVM:',pr
ecision_SVM)
            scores_SVM.append(precision_SVM)
        macc_SVM = np.mean(scores_SVM)
        print ('Mean Precision_SVM:',macc_SVM)
        g_C_SVM.append((gamma,C))
        acc_SVM.append(macc_SVM)
```

```
gamma: 0.001 C: 1  k_fold: 1  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 2  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 3  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 4  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 5  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 6  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 7  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 8  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 9  precision for SVM: 0.647438072616219
9
gamma: 0.001 C: 1  k_fold: 10  precision for SVM: 0.64743807261621
99
Mean Precision_SVM: 0.6474380726162199
gamma: 0.001 C: 10  k_fold: 1  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 2  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 3  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 4  precision for SVM: 0.78791991856124
88
```

```
gamma: 0.001 C: 10  k_fold: 5  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 6  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 7  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 8  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 9  precision for SVM: 0.78791991856124
88
gamma: 0.001 C: 10  k_fold: 10  precision for SVM: 0.7879199185612
488
Mean Precision_SVM: 0.7879199185612488
gamma: 0.001 C: 100  k_fold: 1  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 2  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 3  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 4  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 5  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 6  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 7  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 8  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 9  precision for SVM: 0.8225313878520
529
gamma: 0.001 C: 100  k_fold: 10  precision for SVM: 0.822531387852
0529
Mean Precision_SVM: 0.8225313878520529
gamma: 0.001 C: 1000  k_fold: 1  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 2  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 3  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 4  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 5  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 6  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 7  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 8  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 9  precision for SVM: 0.826603325415
6769
gamma: 0.001 C: 1000  k_fold: 10  precision for SVM: 0.82660332541
56769
Mean Precision_SVM: 0.8266033254156768
gamma: 0.0001 C: 1  k_fold: 1  precision for SVM: 0.49236511706820
496
```

```
gamma: 0.0001 C: 1  k_fold: 2  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 3  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 4  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 5  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 6  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 7  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 8  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 9  precision for SVM: 0.49236511706820
496
gamma: 0.0001 C: 1  k_fold: 10  precision for SVM: 0.4923651170682
0496
Mean Precision_SVM: 0.492365117068205
gamma: 0.0001 C: 10  k_fold: 1  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 2  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 3  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 4  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 5  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 6  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 7  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 8  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 9  precision for SVM: 0.6494740413980
319
gamma: 0.0001 C: 10  k_fold: 10  precision for SVM: 0.649474041398
0319
Mean Precision_SVM: 0.649474041398032
gamma: 0.0001 C: 100  k_fold: 1  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 2  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 3  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 4  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 5  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 6  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 7  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 8  precision for SVM: 0.784526637258
2287
gamma: 0.0001 C: 100  k_fold: 9  precision for SVM: 0.784526637258
```

```
2287
gamma: 0.0001 C: 100  k_fold: 10  precision for SVM: 0.78452663725
82287
Mean Precision_SVM: 0.7845266372582287
gamma: 0.0001 C: 1000  k_fold: 1  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 2  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 3  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 4  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 5  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 6  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 7  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 8  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 9  precision for SVM: 0.81913810654
9033
gamma: 0.0001 C: 1000  k_fold: 10  precision for SVM: 0.8191381065
49033
Mean Precision_SVM: 0.819138106549033
```

In [47]:

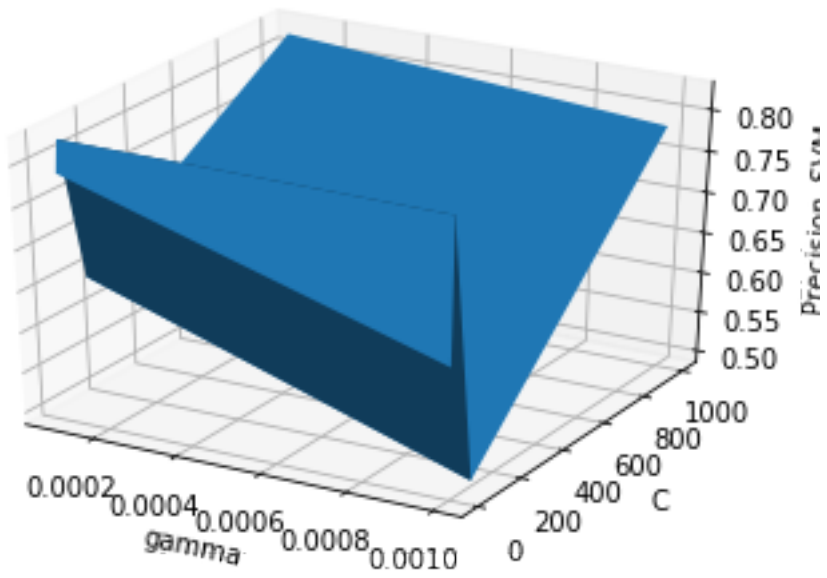
```
import matplotlib.pyplot as plt

fig_4 = plt.figure()
ax_4 = fig_4.add_subplot(111, projection='3d')
X4, Y4 = np.meshgrid(gammas, CP)
zs_4 = np.array(acc_SVM)
Z4 = zs_4.reshape(X4.shape)

ax_4.plot_surface(X4, Y4, Z4)

ax_4.set_xlabel('gamma')
ax_4.set_ylabel('C')
ax_4.set_zlabel('Precision_SVM')

plt.show()
```



In [48]:

```
best_g_C = g_C_SVM[np.argmax(acc_SVM)]
print('Best Gamma:',best_g_C[0], ' Best C:',best_g_C[1])
SVM = svm.SVC(kernel='rbf',gamma=best_g_C[0],C=best_g_C[1])
SVM.fit(X_Train,np.array(Y_Train))
pred_SVM = SVM.predict(X_Test)
f1_Best_SVM = f1_score(np.array(Y_Test),pred_SVM,average='micro')
print ('F1 Accuracy for Best Gamma and C:',f1_Best_SVM)
accu_Best_SVM = accuracy_score(np.array(Y_Test),pred_SVM)
print ('Accuracy for Best Gamma and C:',accu_Best_SVM)
cm_Best_SVM = confusion_matrix(np.array(Y_Test),pred_SVM)
print ('Confusion Matrix for Best Gamma and C:\n',cm_Best_SVM)
```

```
Best Gamma: 0.001  Best C: 1000
F1 Accuracy for Best Gamma and C: 0.8266033254156769
Accuracy for Best Gamma and C: 0.8266033254156769
Confusion Matrix for Best Gamma and C:
 [[454  19  23   0   0   0]
 [ 97 352  22   0   0   0]
 [ 66  36 318   0   0   0]
 [  0   1   1 368 109  12]
 [  1   0   0  87 430  14]
 [  0   1   0  11  11 514]]
```

In [49]:

```
fID = 218187754%4

fID
```

Out[49]:

2

In [50]:

```
# Part 5: Random Forest #
# Build a Random Forest Classifier for this data. #

from sklearn.ensemble import RandomForestClassifier
tree_depth = np.array([300,500,600])
number_of_trees = np.array([200,500,700])

acc_rf = []
a_l1_rf = []
for dep in tree_depth:
    for nt in number_of_trees:
        rf = RandomForestClassifier(n_estimators=nt,max_depth=dep)
        scores_rf = []
        for i in np.arange(1,11):
            rf.fit(X_Train,np.array(Y_Train))
            pred_rf = rf.predict(X_Test)
            #f1_rf = f1_score(np.array(Y_Test),pred_rf,average='micro')
            accu_rf = accuracy_score(np.array(Y_Test),pred_rf)
            print ('Tree Depth:',dep,' No. of trees:',nt,' k_fold:',i,' accu_s
core_rf:',accu_rf)
            scores_rf.append(accu_rf)
        macc_rf = np.mean(scores_rf)
        print ('Mean Accu_rf:',macc_rf)
        a_l1_rf.append((nt,dep))
        acc_rf.append(macc_rf)
```

```
Tree Depth: 300  No. of trees: 200  k_fold: 1  accu_score_rf: 0.83
5765184933831
Tree Depth: 300  No. of trees: 200  k_fold: 2  accu_score_rf: 0.84
08551068883611
Tree Depth: 300  No. of trees: 200  k_fold: 3  accu_score_rf: 0.83
3729216152019
Tree Depth: 300  No. of trees: 200  k_fold: 4  accu_score_rf: 0.83
91584662368511
Tree Depth: 300  No. of trees: 200  k_fold: 5  accu_score_rf: 0.83
9497794367153
Tree Depth: 300  No. of trees: 200  k_fold: 6  accu_score_rf: 0.84
2551747539871
Tree Depth: 300  No. of trees: 200  k_fold: 7  accu_score_rf: 0.83
9497794367153
Tree Depth: 300  No. of trees: 200  k_fold: 8  accu_score_rf: 0.83
91584662368511
Tree Depth: 300  No. of trees: 200  k_fold: 9  accu_score_rf: 0.83
6443841194435
Tree Depth: 300  No. of trees: 200  k_fold: 10  accu_score_rf: 0.8
```

```
388191381065491
Mean Accu_rf: 0.8385476756023074
Tree Depth: 300  No. of trees: 500  k_fold: 1  accu_score_rf: 0.83
6104513064133
Tree Depth: 300  No. of trees: 500  k_fold: 2  accu_score_rf: 0.83
4407872412623
Tree Depth: 300  No. of trees: 500  k_fold: 3  accu_score_rf: 0.83
91584662368511
Tree Depth: 300  No. of trees: 500  k_fold: 4  accu_score_rf: 0.84
08551068883611
Tree Depth: 300  No. of trees: 500  k_fold: 5  accu_score_rf: 0.83
7801153715643
Tree Depth: 300  No. of trees: 500  k_fold: 6  accu_score_rf: 0.83
91584662368511
Tree Depth: 300  No. of trees: 500  k_fold: 7  accu_score_rf: 0.83
7122497455039
Tree Depth: 300  No. of trees: 500  k_fold: 8  accu_score_rf: 0.84
1873091279267
Tree Depth: 300  No. of trees: 500  k_fold: 9  accu_score_rf: 0.83
8140481845945
Tree Depth: 300  No. of trees: 500  k_fold: 10  accu_score_rf: 0.8
408551068883611
Mean Accu_rf: 0.8385476756023074
Tree Depth: 300  No. of trees: 700  k_fold: 1  accu_score_rf: 0.84
0515778758059
Tree Depth: 300  No. of trees: 700  k_fold: 2  accu_score_rf: 0.84
0176450627757
Tree Depth: 300  No. of trees: 700  k_fold: 3  accu_score_rf: 0.83
7801153715643
Tree Depth: 300  No. of trees: 700  k_fold: 4  accu_score_rf: 0.84
1533763148965
Tree Depth: 300  No. of trees: 700  k_fold: 5  accu_score_rf: 0.84
08551068883611
Tree Depth: 300  No. of trees: 700  k_fold: 6  accu_score_rf: 0.84
0176450627757
Tree Depth: 300  No. of trees: 700  k_fold: 7  accu_score_rf: 0.83
5765184933831
Tree Depth: 300  No. of trees: 700  k_fold: 8  accu_score_rf: 0.83
91584662368511
Tree Depth: 300  No. of trees: 700  k_fold: 9  accu_score_rf: 0.83
8479809976247
Tree Depth: 300  No. of trees: 700  k_fold: 10  accu_score_rf: 0.8
449270444519851
Mean Accu_rf: 0.8399389209365458
Tree Depth: 500  No. of trees: 200  k_fold: 1  accu_score_rf: 0.83
9837122497455
Tree Depth: 500  No. of trees: 200  k_fold: 2  accu_score_rf: 0.83
3729216152019
Tree Depth: 500  No. of trees: 200  k_fold: 3  accu_score_rf: 0.83
5765184933831
Tree Depth: 500  No. of trees: 200  k_fold: 4  accu_score_rf: 0.84
0515778758059
Tree Depth: 500  No. of trees: 200  k_fold: 5  accu_score_rf: 0.83
8479809976247
Tree Depth: 500  No. of trees: 200  k_fold: 6  accu_score_rf: 0.83
3389888021717
Tree Depth: 500  No. of trees: 200  k_fold: 7  accu_score_rf: 0.83
```

```
7801153715643
Tree Depth: 500  No. of trees: 200  k_fold: 8  accu_score_rf: 0.83
8479809976247
Tree Depth: 500  No. of trees: 200  k_fold: 9  accu_score_rf: 0.83
6104513064133
Tree Depth: 500  No. of trees: 200  k_fold: 10  accu_score_rf: 0.8
38140481845945
Mean Accu_rf: 0.8372242958941296
Tree Depth: 500  No. of trees: 500  k_fold: 1  accu_score_rf: 0.84
0176450627757
Tree Depth: 500  No. of trees: 500  k_fold: 2  accu_score_rf: 0.83
4068544282321
Tree Depth: 500  No. of trees: 500  k_fold: 3  accu_score_rf: 0.83
91584662368511
Tree Depth: 500  No. of trees: 500  k_fold: 4  accu_score_rf: 0.83
8479809976247
Tree Depth: 500  No. of trees: 500  k_fold: 5  accu_score_rf: 0.83
91584662368511
Tree Depth: 500  No. of trees: 500  k_fold: 6  accu_score_rf: 0.84
08551068883611
Tree Depth: 500  No. of trees: 500  k_fold: 7  accu_score_rf: 0.84
0176450627757
Tree Depth: 500  No. of trees: 500  k_fold: 8  accu_score_rf: 0.83
9497794367153
Tree Depth: 500  No. of trees: 500  k_fold: 9  accu_score_rf: 0.84
1873091279267
Tree Depth: 500  No. of trees: 500  k_fold: 10  accu_score_rf: 0.8
408551068883611
Mean Accu_rf: 0.8394299287410927
Tree Depth: 500  No. of trees: 700  k_fold: 1  accu_score_rf: 0.83
91584662368511
Tree Depth: 500  No. of trees: 700  k_fold: 2  accu_score_rf: 0.83
7122497455039
Tree Depth: 500  No. of trees: 700  k_fold: 3  accu_score_rf: 0.83
9837122497455
Tree Depth: 500  No. of trees: 700  k_fold: 4  accu_score_rf: 0.83
9497794367153
Tree Depth: 500  No. of trees: 700  k_fold: 5  accu_score_rf: 0.83
88191381065491
Tree Depth: 500  No. of trees: 700  k_fold: 6  accu_score_rf: 0.84
08551068883611
Tree Depth: 500  No. of trees: 700  k_fold: 7  accu_score_rf: 0.84
2212419409569
Tree Depth: 500  No. of trees: 700  k_fold: 8  accu_score_rf: 0.84
0176450627757
Tree Depth: 500  No. of trees: 700  k_fold: 9  accu_score_rf: 0.84
1533763148965
Tree Depth: 500  No. of trees: 700  k_fold: 10  accu_score_rf: 0.8
411944350186631
Mean Accu_rf: 0.8400407193756362
Tree Depth: 600  No. of trees: 200  k_fold: 1  accu_score_rf: 0.84
0515778758059
Tree Depth: 600  No. of trees: 200  k_fold: 2  accu_score_rf: 0.83
5765184933831
Tree Depth: 600  No. of trees: 200  k_fold: 3  accu_score_rf: 0.84
08551068883611
Tree Depth: 600  No. of trees: 200  k_fold: 4  accu_score_rf: 0.83
```

```
3050559891415
Tree Depth: 600  No. of trees: 200  k_fold: 5  accu_score_rf: 0.83
9837122497455
Tree Depth: 600  No. of trees: 200  k_fold: 6  accu_score_rf: 0.83
3389888021717
Tree Depth: 600  No. of trees: 200  k_fold: 7  accu_score_rf: 0.84
2212419409569
Tree Depth: 600  No. of trees: 200  k_fold: 8  accu_score_rf: 0.83
47472005429251
Tree Depth: 600  No. of trees: 200  k_fold: 9  accu_score_rf: 0.83
9497794367153
Tree Depth: 600  No. of trees: 200  k_fold: 10  accu_score_rf: 0.8
37461825585341
Mean Accu_rf: 0.8377332880895827
Tree Depth: 600  No. of trees: 500  k_fold: 1  accu_score_rf: 0.83
91584662368511
Tree Depth: 600  No. of trees: 500  k_fold: 2  accu_score_rf: 0.83
91584662368511
Tree Depth: 600  No. of trees: 500  k_fold: 3  accu_score_rf: 0.83
9837122497455
Tree Depth: 600  No. of trees: 500  k_fold: 4  accu_score_rf: 0.83
8479809976247
Tree Depth: 600  No. of trees: 500  k_fold: 5  accu_score_rf: 0.83
8140481845945
Tree Depth: 600  No. of trees: 500  k_fold: 6  accu_score_rf: 0.84
28910756701731
Tree Depth: 600  No. of trees: 500  k_fold: 7  accu_score_rf: 0.83
9497794367153
Tree Depth: 600  No. of trees: 500  k_fold: 8  accu_score_rf: 0.84
01764506277 57
Tree Depth: 600  No. of trees: 500  k_fold: 9  accu_score_rf: 0.84
45877163216831
Tree Depth: 600  No. of trees: 500  k_fold: 10  accu_score_rf: 0.8
37801153715643
Mean Accu_rf: 0.839972853749576
Tree Depth: 600  No. of trees: 700  k_fold: 1  accu_score_rf: 0.83
9837122497455
Tree Depth: 600  No. of trees: 700  k_fold: 2  accu_score_rf: 0.84
28910756701731
Tree Depth: 600  No. of trees: 700  k_fold: 3  accu_score_rf: 0.83
91584662368511
Tree Depth: 600  No. of trees: 700  k_fold: 4  accu_score_rf: 0.83
7801153715643
Tree Depth: 600  No. of trees: 700  k_fold: 5  accu_score_rf: 0.84
0515778758059
Tree Depth: 600  No. of trees: 700  k_fold: 6  accu_score_rf: 0.84
2212419409569
Tree Depth: 600  No. of trees: 700  k_fold: 7  accu_score_rf: 0.83
91584662368511
Tree Depth: 600  No. of trees: 700  k_fold: 8  accu_score_rf: 0.83
91584662368511
Tree Depth: 600  No. of trees: 700  k_fold: 9  accu_score_rf: 0.84
11944350186631
Tree Depth: 600  No. of trees: 700  k_fold: 10  accu_score_rf: 0.8
41873091279267
Mean Accu_rf: 0.8403800475059382
```

In [51]:

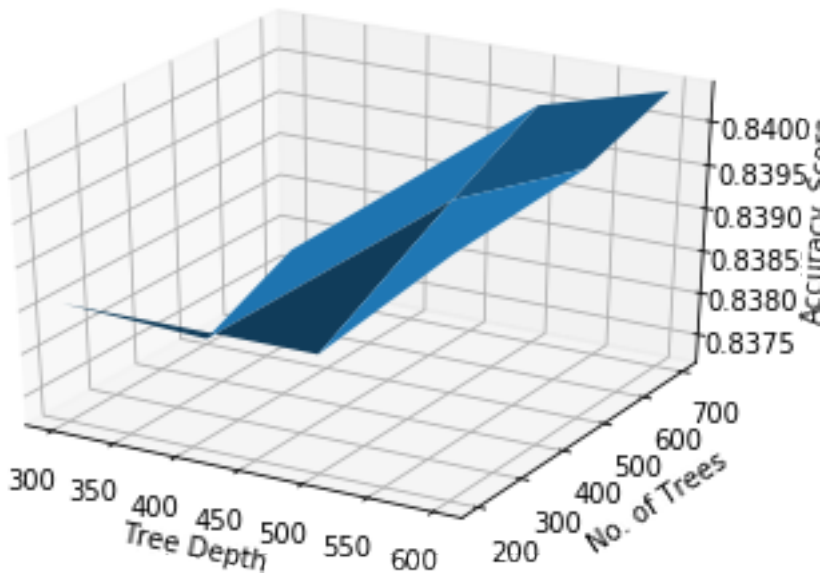
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm

fig_5 = plt.figure()
ax_5 = fig_5.add_subplot(111, projection='3d')
X5, Y5 = np.meshgrid(tree_depth, number_of_trees)
zs_5 = np.array(acc_rf)
Z5 = zs_5.reshape(X5.shape)

ax_5.plot_surface(X5, Y5, Z5)

ax_5.set_xlabel('Tree Depth')
ax_5.set_ylabel('No. of Trees')
ax_5.set_zlabel('Accuracy_Score')

plt.show()
```



In [52]:

```
best_value_5 = a_l1_rf[np.argmax(acc_rf)]
print('Best No. of trees:',best_value_5[0], ' Best depth:',best_value_5[1])
rf_best5 = RandomForestClassifier(n_estimators=best_value_5[0],max_depth=best_
value_5[1])
rf_best5.fit(X_Train,np.array(Y_Train))
pred_rf_best5 = rf_best5.predict(X_Test)
f1_rf_best5 = f1_score(np.array(Y_Test),pred_rf_best5,average='micro')
print ('F1 Accuracy for best value:',f1_rf_best5)
accu_rf_best5 = accuracy_score(np.array(Y_Test),pred_rf_best5)
print ('Accuracy for best value:',accu_rf_best5)
cm_rf_best5 = confusion_matrix(np.array(Y_Test),pred_rf_best5)
print ('Confusion Matrix for best value:\n',cm_rf_best5)
```

```
Best No. of trees: 700  Best depth: 600
F1 Accuracy for best value: 0.837122497455039
Accuracy for best value: 0.837122497455039
Confusion Matrix for best value:
 [[471  13  12   0   0   0]
 [ 58 395  18   0   0   0]
 [ 94  48 278   0   0   0]
 [  0   3   0 337 132  19]
 [  0   0   0  68 457   7]
 [  0   2   0   0   6 529]]
```

In [32]:

```
# Part 6: Discussion #
print ('Write a brief discussion about which classification method achieved th
e best performance and your thoughts on the reason behind this.')
print ('--Random Forest Classifier achieved the best performance because it ha
s the highest accuracy and F1-score.')
```

```
Write a brief discussion about which classification method achieve
d the best performance and your thoughts on the reason behind this
.
--Random Forest Classifier achieved the best performance because i
t has the highest accuracy and F1-score.
```

In [33]:

```
print ('Which method performed the worst and why?')
print ('--K-Nearest Neighbour Classifier performed the worst as it has the low
est accuracy and F1-score. ')
```

```
Which method performed the worst and why?
--K-Nearest Neighbour Classifier performed the worst as it has the
lowest accuracy and F1-score.
```

In [ ]: