In [26]:
```python
import os, sys
#used for searching files of specific pattern
from glob import glob
import joblib
```

In [2]:
```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

In [3]:
```python
from skimage.io import imread_collection
import matplotlib.image as map_img
from matplotlib.image import imread
```

In [4]:
```python
sns.set()
```

In [5]:
```python
#Environment has been setted
```

In [6]:
```python
very_mild=glob(r"Very_Mild_Demented\*")
```

In [7]:
```python
mild = glob(r"Mild_Demented\*")
```

In [8]:
```python
moderate = glob(r"Moderate_Demented\*")
```

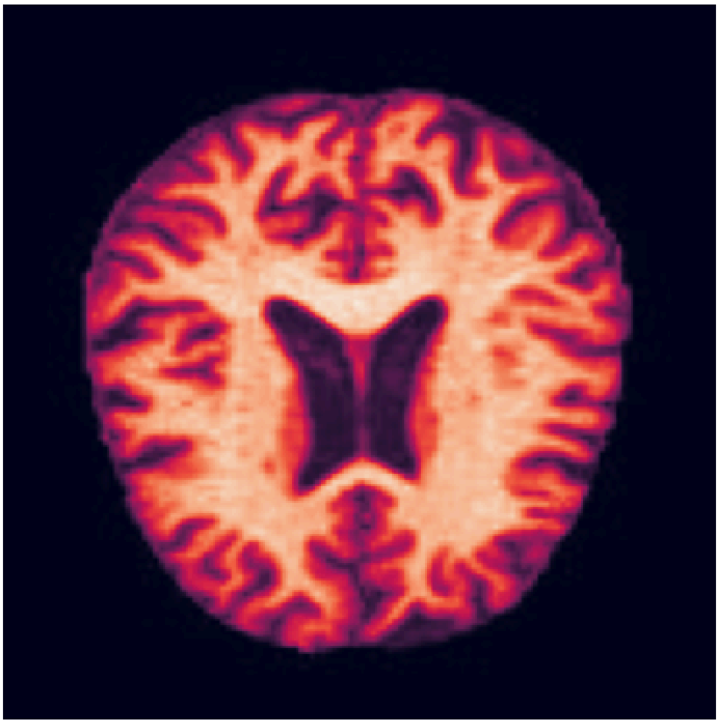
In [9]:
```python
non_demented = glob(r"Non_Demented\*")
```

In [10]:
```python
print(non_demented[1])
def view_image(directory):
    img=map_img.imread(directory)
    plt.imshow(img)
    plt.title(directory)
    plt.axis('off')
    print(f'Image shape:{img.shape}')
    return img
print('Non Demented image data')
view_image(non_demented[1])
```

```
F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Non_Demented\non_10.jpg
Non Demented image data
Image shape:(128, 128)
```

Out[10]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

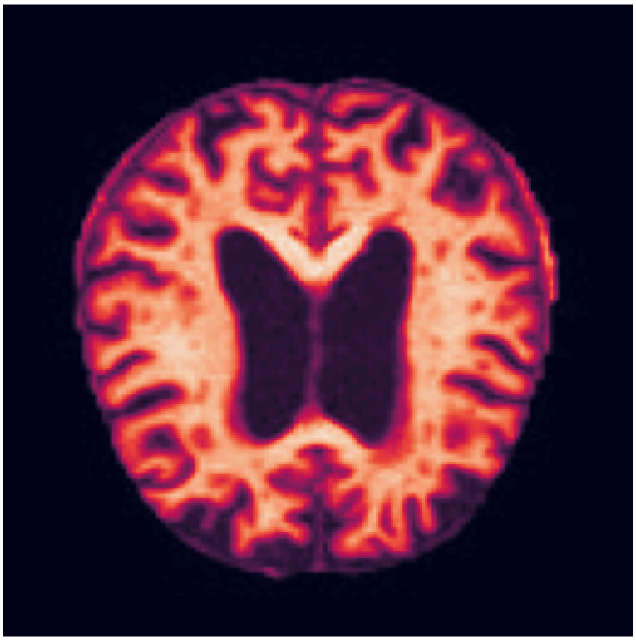
F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Non_Demented\non_10.jpg



In [11]:
```python
print(very_mild[1])
print('Very Mild Demented image data')
view_image(very_mild[1])
```

```
F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Very_Mild_Demented\verymild_
10.jpg
Very Mild Demented image data
Image shape:(128, 128)
```

Out[11]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

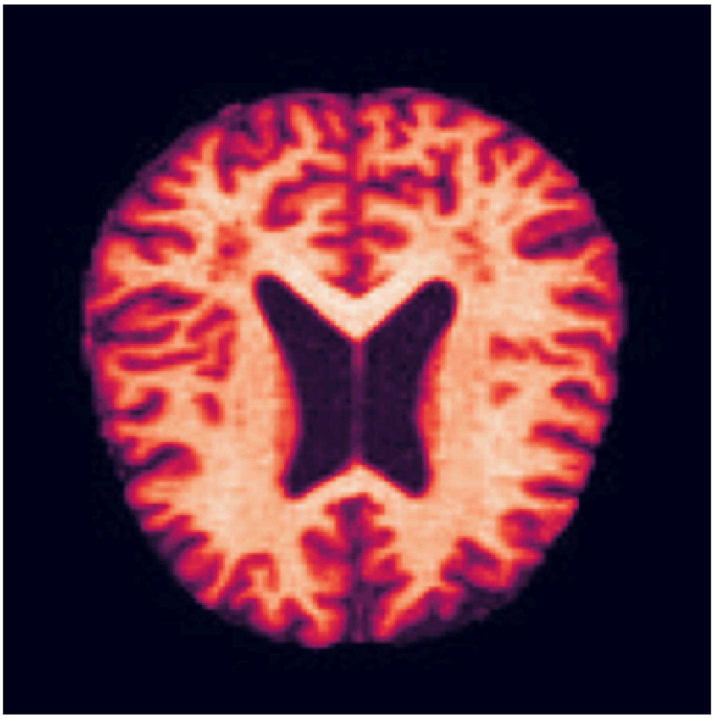


In [12]:
```python
print(mild[1])
print('Mild Demented image data')
view_image(mild[1])
```

```
F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Mild_Demented\mild_10.jpg
Mild Demented image data
Image shape:(128, 128)
```

Out[12]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Mild_Demented\mild_10.jpg

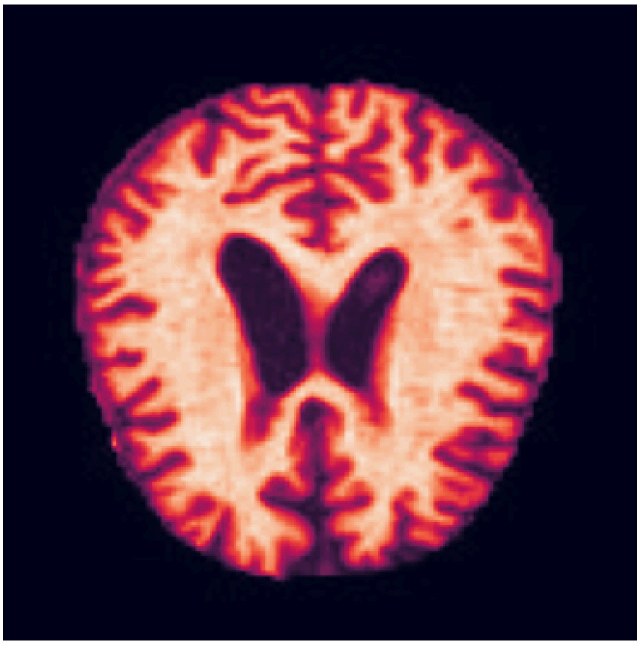


```
In [13]: print(moderate[11])
         print('Moderately Demented image data')
         view_image(moderate[11])
```

```
F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Moderate_Demented\moderate_
2.jpg
Moderately Demented image data
Image shape:(128, 128)
```

```
Out[13]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

F:\Tejaswini\5th_Sem\PBL\Dataset(approx_4000)\Moderate_Demented\moderate_2.jpg



PCA for Alzheimer Detection

```
In [14]: #normalizing pixel , resizing and reshaping
         import matplotlib.image as mpimg
         from skimage.transform import resize
         def extract_feature(dir_path):
             img = mpimg.imread(dir_path)
             img = img/255.0
             img = resize(img,(128,128,3))
             img = np.reshape(img,(128,384))
             return img
```

```
In [15]: non_alz = [extract_feature(filename) for filename in non_demented]
         very_mild_alz = [extract_feature(filename) for filename in very_mild]
         mild_alz = [extract_feature(filename) for filename in mild]
         moderate_alz = [extract_feature(filename) for filename in moderate]
```

In [16]:
```
#concatenated all data
all_data = very_mild_alz + mild_alz + moderate_alz
data = np.concatenate((np.array(non_alz),np.array(all_data)))
```

Done with image reshaping adn clubbing all stages of alz in one

In [17]:
```
data = data.reshape(data.shape[0], np.product(data.shape[1:]))
```

In [18]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data)
```

Out[18]:
```
StandardScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:
```
#standardize to mean 0 and unit variance
x = scaler.transform(data)
```

In [20]:
```
from sklearn.model_selection import train_test_split

all_data = very_mild_alz+mild_alz+moderate_alz
y = [0]*len(non_alz) + [1]*len(all_data)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, y, test_size=0.2)
```

In [21]:
```
from sklearn import decomposition, preprocessing, svm

scale = preprocessing.StandardScaler()
#Compressing the images into two dimensions using PCA
pca = decomposition.PCA(200)
X_proj = pca.fit_transform(x_train)
```

In [22]:
```
#let's first see which principal component works better
#scree plot but cumulative
# Getting the cumulative variance
var_cumu = np.cumsum(pca.explained_variance_ratio_)*100 #100 is multiplied
```

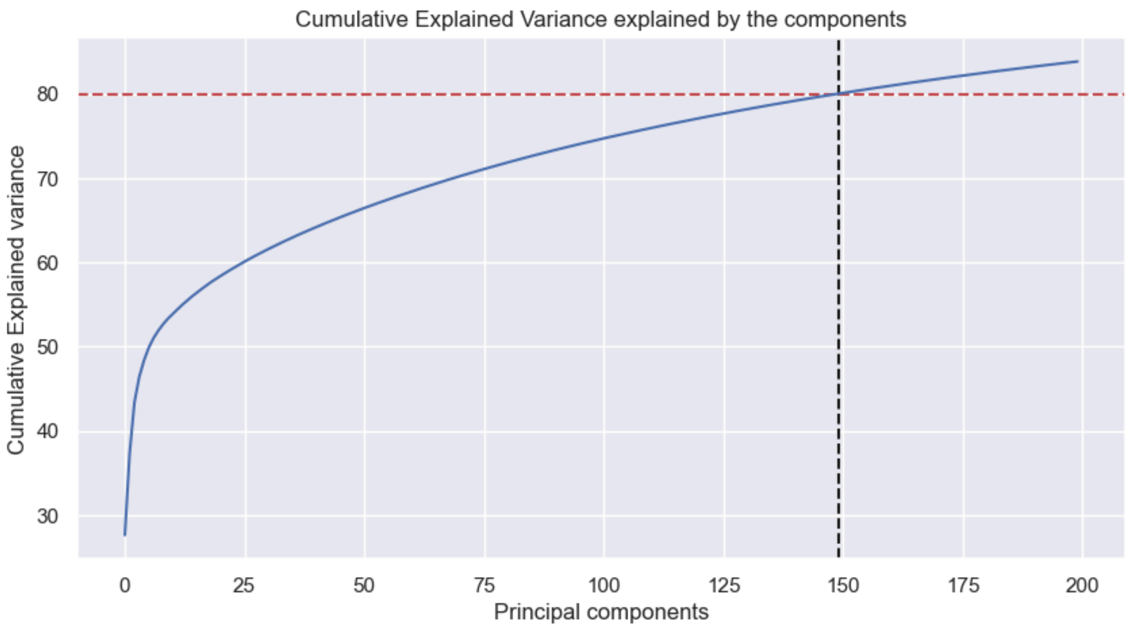
In [23]:
```
# How many PCs explain 90% of the variance?
k = np.argmax(var_cumu>80)
print("Number of components explaining 80% variance: "+ str(k)) #I guess, I
#print("\n")
```

```
Number of components explaining 80% variance: 149
```

In [24]:
```
plt.figure(figsize=[10,5])
plt.title('Cumulative Explained Variance explained by the components')
plt.ylabel('Cumulative Explained variance')
plt.xlabel('Principal components')
plt.axvline(x=k, color="k", linestyle="--")
plt.axhline(y=80, color="r", linestyle="--")
ax = plt.plot(var_cumu)

print(X_proj)
```

```
[[-13.43688481 -11.30127656   9.03446014 ...  -2.20817952   0.52126778
   -1.3568788 ]
 [-12.29592859  -9.8925087    5.62431683 ...   0.46095089   0.50109636
   -0.92996635]
 [ -5.90365079  15.94549151 -12.66399215 ...   0.69343918   0.11955199
   -0.65691908]
 ...
 [ 23.96510631 -14.76898223  -3.13112791 ...   0.44502832   1.69154231
   -0.32852584]
 [-12.62645057 -12.09336746   1.99751597 ...   0.45347952   0.50108514
   -0.35283175]
 [-11.73080273 -14.01457856   5.2077089  ...   0.35230983   1.3674861
    0.80816671]]
```



In [28]:
```
joblib.dump(scaler, "PCA_graph.model")
```

Out[28]: ['PCA_graph.model']

In [29]:
```
load_scaler = joblib.load("PCA_graph.model")
```

In [30]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=1)
X_train_LDA = lda.fit_transform(x_train, y_train)
X_test_LDA = lda.transform(x_test)
accuracy = lda.score(x_test, y_test)
print(accuracy*100, '% accuracy (testing data)' )
accuracy_train = lda.score(x_train, y_train)
print(accuracy_train*100, '% accuracy (training data)')
```

```
90.25893958076449 % accuracy (testing data)
99.96916435399321 % accuracy (training data)
```

In [32]:
```
joblib.dump(lda, "lda_classification.model")
```

Out[32]:
```
['lda_classification.model']
```

In [33]:
```
load_lda = joblib.load("lda_classification.model")
```

In [34]:
```
# SVM for detection
#List where arrays shall be stored
resized_image_array=[]
#List that will store the answer if an image is female (0) or male (1)
resized_image_array_label=[]

width = 256
height = 256
new_size = (width,height) #the data is just black to white
```

In [35]:
```
# #Iterate over pictures and resize them to 256 by 256
from PIL import Image
from sklearn import decomposition, preprocessing, svm
import sklearn.metrics as metrics
from time import sleep
from tqdm.notebook import tqdm
def resizer(image_directory):
    for file in image_directory: #tried with os.listdir but could work with
        img = Image.open(file) #just putting image_directory or file does n
        #preserve aspect ratio
        img = img.resize(new_size)
        array_temp = np.array(img)
        shape_new = width*height
        img_wide = array_temp.reshape(1, shape_new)
        resized_image_array.append(img_wide[0])
        if image_directory == non_demented:
            resized_image_array_label.append(0)
        else:
            resized_image_array_label.append(1)

ALZ = very_mild + mild + moderate
resizer(non_demented)
resizer(ALZ)
```

In [36]:
```
print(len(non_demented))
print(len(ALZ)) #data are well transformed. Let's conduct SVM
print(len(resized_image_array))
print(resized_image_array[1])
print(len(resized_image_array_label))

#split the data to test and training
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(resized_image_array, re
```

```
1650
2404
4054
[0 0 0 ... 0 0 0]
4054
```

In [50]:
```
clf = svm.SVC(kernel = 'linear')
clf.fit(x_train, y_train)
#store predictions and ground truth
y_pred = clf.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred)) #The

#Now, use the SVM model to predict Test data
y_pred = clf.predict(x_test)
y_true = y_test

#assess the performance of the SVM with linear kernel on Testing data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred))
```

```
Accuracy :  1.0
Precision :  1.0
Recall :  1.0
f1 :  1.0
Confusion matrix : [[1329    0]
 [   0 1914]]
Accuracy :  0.9876695437731196
Precision :  0.9878048780487805
Recall :  0.9918367346938776
f1 :  0.989816700610998
Confusion matrix : [[315   6]
 [  4 486]]
```

In [51]:
```
joblib.dump(clf, "clf_linear.model")
```

Out[51]:
```
['clf_linear.model']
```

In [52]:
```
load_linear = joblib.load("clf_linear.model")
```

In [53]:
```
#Train a SVM using RBF kernel
clf = svm.SVC(kernel = 'rbf')
clf.fit(x_train, y_train)

#store predictions and ground truth
y_pred = clf.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred))

#Now, use the SVM model to predict Test data
y_pred = clf.predict(x_test)
y_true = y_test

#assess the performance of the SVM with linear kernel on Testing data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred))
```

```
Accuracy :  0.8794326241134752
Precision :  0.883241066935078
Recall :  0.9169278996865203
f1 :  0.89976928992566
Confusion matrix : [[1097  232]
 [ 159 1755]]
Accuracy :  0.8335388409371147
Precision :  0.8420038535645472
Recall :  0.8918367346938776
f1 :  0.8662041625371654
Confusion matrix : [[239  82]
 [ 53 437]]
```

In [54]:
```
joblib.dump(clf, "clf_rbf.model")
```

Out[54]:
```
['clf_rbf.model']
```

In [55]:
```
load_clfd = joblib.load("clf_rbf.model")
```

In [56]:
```
#Train a SVM using polynomial kernel with degree of 2
clf = svm.SVC(kernel = 'poly', degree = 2)
clf.fit(x_train, y_train)

#store predictions and ground truth
y_pred = clf.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred))

# SVM model to predict Test data
y_pred = clf.predict(x_test)
y_true = y_test

#assess the performance of the SVM with linear kernel on Testing data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
print('Precision : ', metrics.precision_score(y_true, y_pred))
print('Recall : ', metrics.recall_score(y_true, y_pred))
print('f1 : ', metrics.f1_score(y_true, y_pred))
print('Confusion matrix :', metrics.confusion_matrix(y_true, y_pred))
```

```
Accuracy :  0.9130434782608695
Precision :  0.9125379170879676
Recall :  0.9430512016718914
f1 :  0.9275436793422406
Confusion matrix : [[1156  173]
 [ 109 1805]]
Accuracy :  0.8643649815043156
Precision :  0.8696498054474708
Recall :  0.9122448979591836
f1 :  0.8904382470119522
Confusion matrix : [[254  67]
 [ 43 447]]
```

In [57]:
```
joblib.dump(clf, "clf_poly.model")
```

Out[57]:
```
['clf_poly.model']
```

In [58]:
```
load_clfd1 = joblib.load("clf_poly.model")
```

In [59]:

```
#List where arrays shall be stored
resized_image_array=[]
#List that will store the answer if an image is female (0) or male (1)
resized_image_array_label=[]

width = 256
height = 256
new_size = (width,height) #the data is just black to white

#Iterate over pictures and resize them to 256 by 256
def resizer(image_directory):
    for file in image_directory: #tried with os.listdir but could work with
        img = Image.open(file) #just putting image_directory or file does n
        #preserve aspect ratio
        img = img.resize(new_size)
        array_temp = np.array(img)
        shape_new = width*height
        img_wide = array_temp.reshape(1, shape_new)
        resized_image_array.append(img_wide[0])
        if image_directory == non_demented:
            resized_image_array_label.append(0)
        elif image_directory == very_mild:
            resized_image_array_label.append(1)
        elif image_directory == mild:
            resized_image_array_label.append(2)
        else:
            resized_image_array_label.append(3)

resizer(non_demented)
resizer(very_mild)
resizer(mild)
resizer(moderate)

#split the data to test and training
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(resized_image_array, re

#train SVM model
#from sklearn import svm
clf_c = svm.SVC(kernel = 'linear')
clf_c.fit(x_train, y_train)
#store predictions and ground truth
y_pred = clf_c.predict(x_train)
y_true = y_train
print(y_pred)
#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true, y_pred))
```

```
[0 0 2 ... 1 0 0]
Accuracy :  1.0
```

In [60]:
```python
#Now, use the SVM model to predict Test data
y_pred = clf_c.predict(x_test)
y_true = y_test
print(y_pred)
#assess the performance of the SVM with linear kernel on Testing data
# print("Accuracy : ", metrics.accuracy_score(y_test, y_pred))
```

```
[0 1 1 1 2 1 2 1 1 0 0 0 1 0 1 0 2 2 1 0 1 0 0 2 1 0 0 1 2 0 1 1 2 0 0 2 2
 1 0 1 1 1 1 0 2 0 0 2 1 2 0 2 0 1 1 0 1 1 2 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1
 3 1 1 3 1 1 1 1 0 2 1 1 1 1 0 0 2 0 0 1 1 0 1 1 0 0 0 2 0 2 0 0 1 1 2 1 1
 1 2 0 0 0 0 0 1 0 1 0 1 0 1 3 1 1 1 2 1 0 1 2 0 2 0 2 2 1 0 2 1 0 1 0 1 1
 2 1 1 1 0 1 1 0 1 1 0 0 0 0 1 2 1 2 0 2 0 1 1 1 1 1 0 0 2 2 0 0 1 1 0 2 2
 2 1 1 1 0 1 1 1 0 1 1 0 0 1 1 2 2 1 2 1 2 0 2 0 0 2 0 3 0 1 2 2 0 1 0 2 1
 1 0 0 0 2 2 0 1 0 1 2 2 0 1 1 0 1 2 1 0 2 0 3 0 2 0 1 0 1 2 1 0 0 2 2 0 0
 3 0 0 0 0 1 0 1 0 0 0 2 0 2 0 0 1 1 0 1 1 0 0 2 1 0 1 1 2 0 2 1 0 2 2 1 0
 0 1 2 2 0 1 1 1 1 1 3 0 2 0 0 0 2 0 1 1 0 2 0 0 1 2 1 1 0 0 1 0 2 1 1 0 1
 1 1 1 0 0 0 1 0 2 1 0 0 1 2 1 0 1 0 0 0 0 1 0 1 0 0 2 1 1 2 2 0 1 1 1 0 2
 1 1 2 1 1 1 0 2 2 1 3 0 1 1 1 1 2 2 2 1 0 0 0 1 0 1 1 0 3 0 1 1 0 0 0 0 1
 2 1 0 0 1 1 1 2 0 1 2 1 0 0 2 1 0 1 1 0 0 1 0 2 0 0 2 0 1 3 1 0 0 0 0 0 2
 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 2 0 2 1 1 1 0 1 1 0 2 0 1 2 0 1 1 2 0 0 0 1
 0 1 0 0 0 0 2 0 2 1 0 0 1 0 0 1 1 1 0 0 0 2 1 1 1 1 1 1 1 1 0 1 0 2 1 2 0
 2 0 1 0 0 3 2 0 1 0 0 0 0 0 1 2 1 0 0 1 1 2 1 0 1 1 1 2 2 2 2 1 1 0 0 2 1
 0 0 1 1 1 0 2 0 0 1 1 0 1 2 0 1 1 1 0 1 2 0 1 0 0 0 1 1 1 0 0 0 0 1 2 2 0
 1 0 1 1 1 2 1 1 1 0 0 1 0 1 1 1 2 0 2 1 2 2 0 0 2 0 0 0 0 0 2 0 0 1 0 1 2
 0 1 1 0 0 0 1 1 1 2 0 2 1 0 0 2 0 0 0 0 2 0 0 1 2 1 1 1 0 0 0 1 1 0 0 2 2
 0 1 0 0 0 0 0 0 2 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 2 0 2 2 1 0 0 0 1 1 0 2
 1 1 1 1 0 1 1 2 1 0 0 0 0 0 1 0 0 2 0 2 1 0 1 1 2 1 1 2 0 1 1 1 1 0 0 1 2
 1 2 1 0 2 0 0 0 2 2 2 0 2 2 3 2 0 1 0 1 1 0 2 0 2 0 1 2 1 0 2 1 0 0 1 0 2
 1 0 0 1 2 0 2 0 1 2 0 1 0 2 1 1 1 3 1 1 1 2 0 1 0 1 0 2 2 1 1 0 0 3]
```

In [61]:
```python
print("Accuracy : ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy :  0.9889025893958077
```

In [63]:
```python
joblib.dump(clf_c, "clf_l1.model")
```

Out[63]:
```
['clf_l1.model']
```

In [65]:
```python
load_clfc = joblib.load("clf_l1.model")
```

In [66]:
```python
clf_c1 = svm.SVC(kernel = 'poly', degree = 2)
clf_c1.fit(x_train,y_train)

#store predictions and ground truth
y_pred = clf_c1.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true,y_pred))

#Now, use the SVM model to predict Test data
y_pred = clf_c1.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Testing data
print('Accuracy : ', metrics.accuracy_score(y_true,y_pred))
```

```
Accuracy :  0.8825161887141536
Accuracy :  0.8825161887141536
```

In [69]:
```
joblib.dump(clf_c1, "clf_p1.model")
```

Out[69]:
```
['clf_p1.model']
```

In [70]:
```
load_clfc1 = joblib.load("clf_p1.model")
```

In [71]:
```
#Train a SVM using RBF kernel
clf_c2 = svm.SVC(kernel = 'rbf')
clf_c2.fit(x_train, y_train)

#store predictions and ground truth
y_pred = clf_c2.predict(x_train)
y_true = y_train

#assess the performance of the SVM with linear kernel on Training data
print('Accuracy : ', metrics.accuracy_score(y_true,y_pred))

#Now, use the SVM model to predict Test data
y_pred = clf_c2.predict(x_test)
y_true = y_test

#assess the performance of the SVM with linear kernel on Testing data
print('Accuracy : ', metrics.accuracy_score(y_true,y_pred))
```

```
Accuracy :  0.7980265186555658
Accuracy :  0.7163995067817509
```

In [72]:
```
joblib.dump(clf_c2, "clf_r1.model")
```

Out[72]:
```
['clf_r1.model']
```

In [73]:
```
load_clfc2 = joblib.load("clf_r1.model")
```