Politecnico di Torino

01SQIOV

Artificial Intelligence and Machine Learning

Homework 1:
PCA and Naive Bayes Classifier

Prof. Barbara Caputo
Prof. Paolo Russo

Nour Saeed
Politecnico di Torino
Department of Computer Engineering, Cinema, & Mechatronics
S250409@studenti.polito.it

January 5, 2019

Abstract

This report is a required deliverable for the course of AIML. It deals with PCA and Naive Bayesian Classifier.

Principal Component Analysis

Setup

For this homework, I used PyCharm as my IDE and created a virtual environment in which I added all required libraries.
After downloading the PACS dataset, I added it to my project directory. The dataset folder is called Images and has 4 sub folders each containing one class of images.
Let's first import the required libraries

```
from PIL import Image
import numpy as np
import random
import os
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.decomposition import PCA
```

And we load the images:

```
data=[] # data
y=[]    # Label


# Load Images
directories=os.listdir('./Images/')
print("Start Loading")
for j in directories:
    allfiles = os.listdir('./Images/' + j + '/')
    imlist = [filename for filename in allfiles if filename[-4:]
    in ['.jpg']]
    for i in range(len(imlist)):
        img_data=np.asarray(Image.open('./Images/' + j + '/' +
   imlist[i]))

        # Data matrix update
        data.append(img_data.ravel())

        # Label vector update
        if (j == 'dog'):
            y.append(1)
        if(j=='guitar'):
            y.append(2)
        if(j=='house'):
```

```
            y.append(3)
        if(j=='person'):
            y.append(4)

shape=img_data.shape
print("Done loading images")
```

Images are converted into vectors through the ravel() function. We add a label vector where 1 is dog 2 is guitar 3 is house and 4 is person. We also save the shape of the image so that we could use it later.

Standardization

We now have to standardize the data. This can be easily done through sklearn's StandardScaler. This centers the data around the mean and provides a unit variance.

```
# Standardize Data
print("Standardizing the data")
scaler = preprocessing.StandardScaler()
data_standardized= scaler.fit_transform(data)
```

```
In[4]: data_standardized.mean(axis=0)
Out[4]:

array([2.57179538e-16, 1.03566343e-16, 1.68116568e-16, ...,
       2.06315594e-16, 7.35382316e-17, 1.34820091e-17])
In[5]: data_standardized.var(axis=0)
Out[5]:
array([1., 1., 1., ..., 1., 1., 1.])
```

PCA

```
imgNum=random.randint(0,len(data)-1)
```

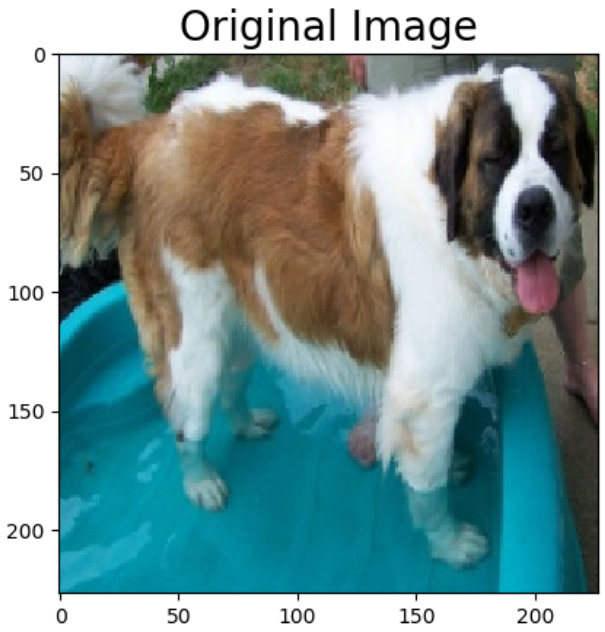
This selects a random image index from our dataset .
The following function is a function that uses sklearn's PCA to get the new dataset.

```
def PrincipleComponent(compNum, data):
    pca = PCA(compNum)
    X_t = pca.fit_transform(data)
    approximation = pca.inverse_transform(X_t)
    return X_t,approximation
```

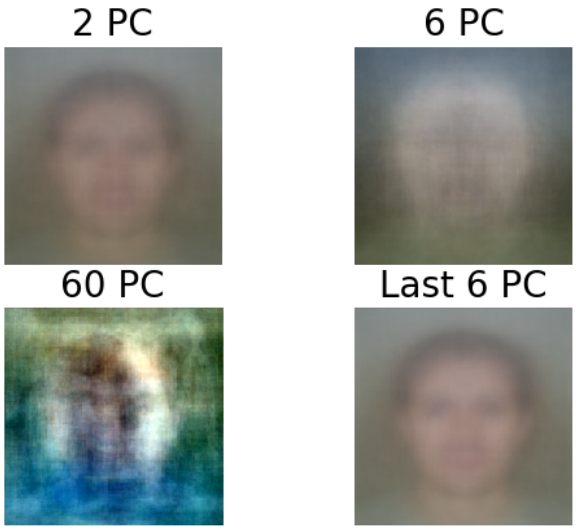
For 2 components, the PCs are:

```
array([[ 0.00339956,  0.00361869,  0.00358235, ...,  0.00245523,
         0.00258395,  0.00228174],
       [ 0.00103133,  0.00137747,  0.00178421, ..., -0.00186904,
        -0.00218749, -0.00305935]])
```

They can be viewed through pca.components_ after fitting and training. A random image is chosen:

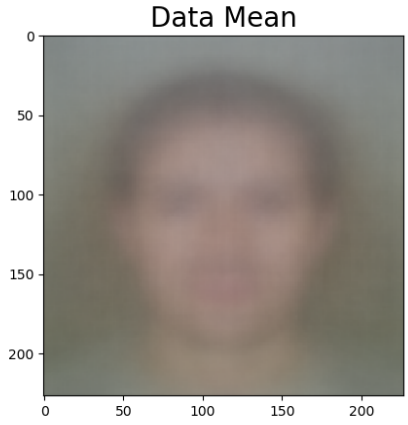


Testing out 60, 6, 2, and the last 6 PCs, we get:



After projecting the image using a certain number of PCs, we get these. All images have a person shape.

Let's check the mean:



```
mean=scaler.mean_/255
plt.imshow(mean.reshape(shape))
plt.title('Data Mean', fontsize=20)
plt.savefig('./Run/mean.png')
plt.show()
```

Looking at the dataset, we have more persons than other classes. This explains why our output is like this. A possible fix would be adjusting the dataset.
These projections now make sense. With *2 PCs* the data would be closer to that of the most variance. Considering that we have 200 images more of people, this could be the reason. As we increase the PCs to 6 we notice a slight change. Increasing further to 60 PCs, we can see the dog somewhat reappearing. For the case of the *last 6 PCs*, the projection looks like the mean and this is somewhat expected considering that for the last 6 PCs aren't important and their effect is small.

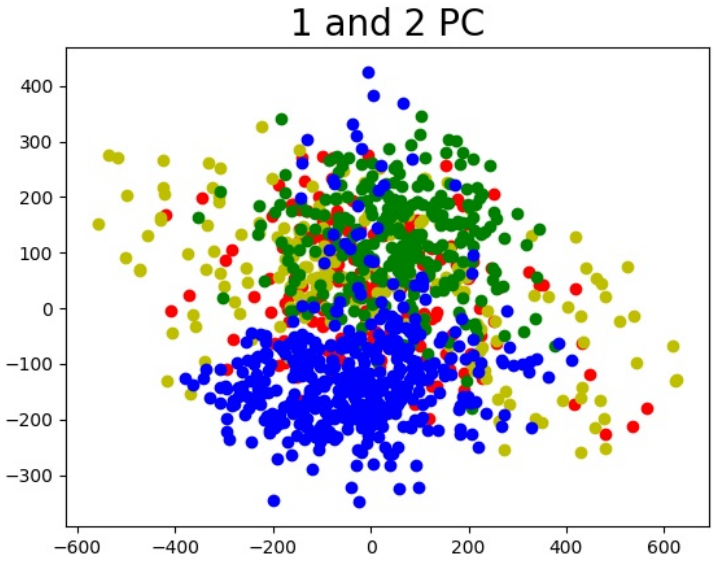
Visualizing two PCs

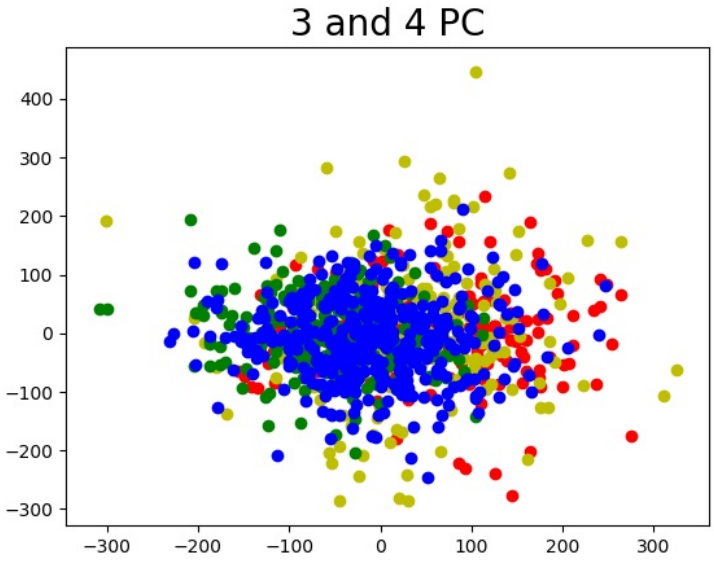
The images below show the dataset projected on the first 2 PCs, the 3^{rd} and 4^{th} PCs, and the 11^{rd} and 12^{th} PCs.

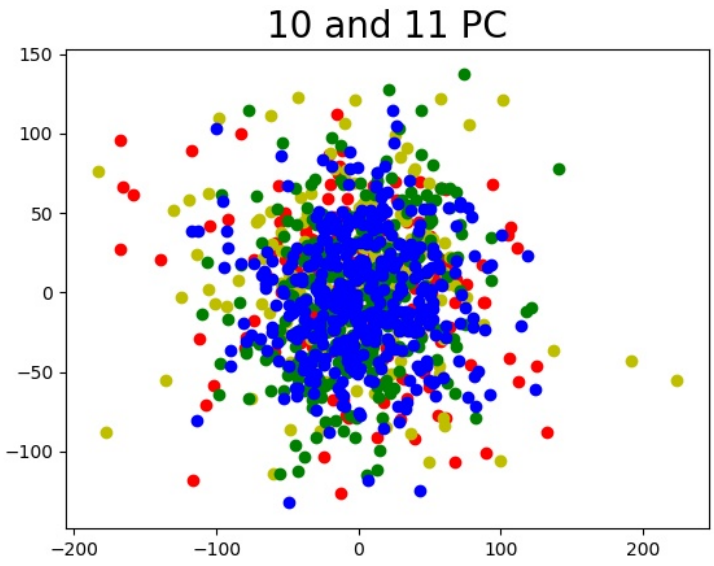
```
color=['r','y','g','b']

for j in [0,2,9]:
    for i in range(len(y)):
        plt.scatter(X_t[i,j],X_t[i,j+1],c=color[y[i]-1])
    plt.title('%i and %i PC' %(j+1,j+2), fontsize = 20)
    plt.savefig('./Run/%i.jpg' %j)
    plt.show()
```

Red dots corresponds to dogs, yellow to guitars, green to houses and blue to persons.
With the first 2 PCs, the data is most disbursed. As we choose other PCs, the data gets less and less disbursed and becomes more clustered. This is due to the PCA. In PCA, the most principle component is selected based on maximizing the variance. The second is selected by maximizing the variance of the data minus the projections of the data on the previous PCAs...

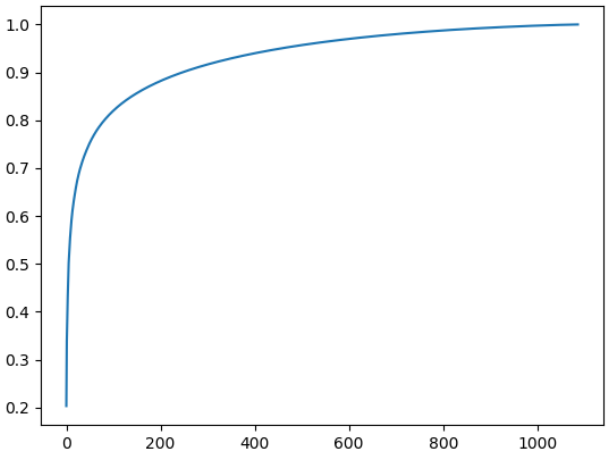

1 and 2 PC
400
300
200
100
0
−100
−200
−300
−600
−400
−200
0
200
400
600


3 and 4 PC
400
300
200
100
0
−100
−200
−300
−300
−200
−100
0
100
200
300



Component Selection:

From the given, we apparently need more PCs. Plotting the explained variance:



We need around 200 PCs to get around 88% of the variance. With sklearn,

we can set the n$_components of the PCA() to a values between 0 and 1. This value would represent the per

Naive Bayes Classifier

The python file is named (NB.py)

The Theory

$$\hat{y} = argmax p(y|x_1, ..., x_d)$$

From Bayes theorem, we get:

$$p(y|x_1, ..., x_d) = \frac{p(x_1, ..., x_d|y)p(y)}{p(x_1, ..., x_d)}$$

Being naive, we consider that all x_i are independent. So:

$$p(x_1, ..., x_d) = \prod_{i=1}^{d} p(x_i) = constant$$

$$p(x_1, ..., x_d|y) = \prod_{i=1}^{d} p(x_i|y)$$

So we have:

$$p(y|x_1, ..., x_d) \alpha p(y) \prod_{i=1}^{d} p(x_i|y) = f_{NB}$$

Classification

Classification can be done through sklearn. We first split the data. Since we have around 1000 images, I set the test size to 0,1.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_t, y, test_size=0.1,random_state=6)
```

For the entire dataset, I get the following results:

```
Our accuracy without dimension reduction is: 0.7798165137614679
The confusion matrix is:
[[ 9  4  2  2]
 [ 3 12  0  0]
 [ 4  5 26  0]
 [ 1  2  1 38]]
```

Using the first 2 PCs, I get:

```
Our accuracy with dimension reduction using 1st and 2nd PCs is: 0.6238532110091743
The confusion matrix is:
[[ 4  3  6  4]
 [ 3  8  4  0]
 [ 7  6 20  2]
 [ 3  1  2 36]]
```

Using the 3rd and 4th PC, the results are:

```
Our accuracy with dimension reduction using 3rd and 4th PCs is: 0.5045871559633027
The confusion matrix is:
[[ 8  2  2  5]
 [ 4  7  3  1]
 [ 3  2 26  4]
 [ 8  3 17 14]]
```

By using only the first two PC, we have lost accuracy but we still managed to get more than half right. We have a 20% decrease in accuracy but we have 99% reduction in dimensions.
If we choose the 3rd and 4th PCs (not as significant as the 1st two PCs), the accuracy decreases more. This is due to the PCA.

Decision Boundaries

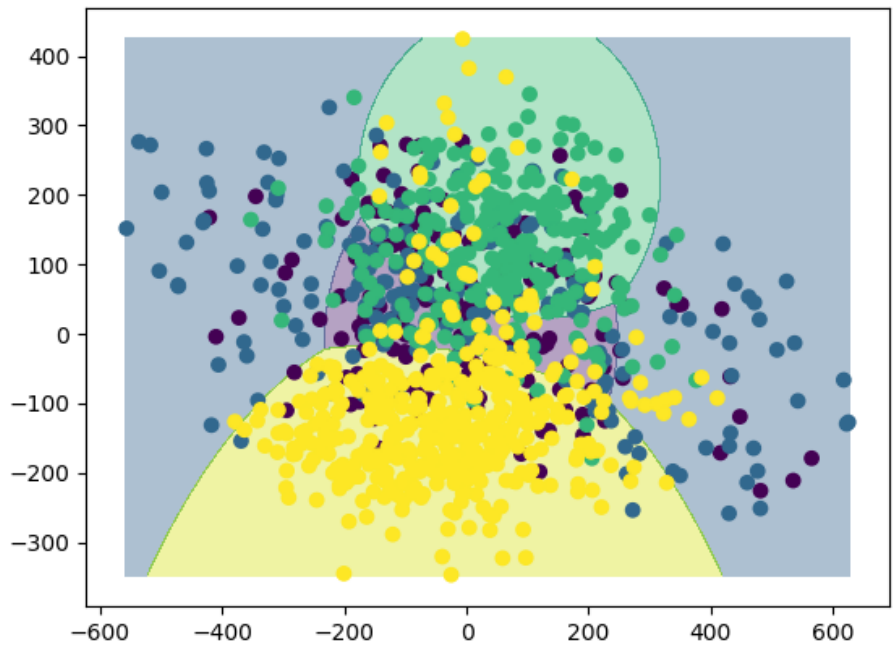
The decision boundaries can be plotted as show using the data projected on the 1st two PCs.

```
x_min, x_max = X_t[:,0].min() - 1, X_t[:,0].max() + 1
y_min, y_max = X_t[:,1].min() - 1, X_t[:,1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.2),
                     np.arange(y_min, y_max, 0.2))
```

```
Z = gnb.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

#['dog':'purple','guitar':'blue', 'house':'green','person':'
    yellow']
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_t[:,0],X_t[:,1],c=y)
plt.savefig('./Run/Descision.png')
plt.show()
```

The output is as follows:



Purple is dogs. Blue is guitar. Green is house. Yellow is person.
As you can see, these boundaries aren't accurate as part of the data falls outside the boundary.