Machine_Learning_Homework1

January 19, 2019

DATA IMPORT

In this homework we are going to use a statistical proceudre to convert a set of observations of possibly correlated varabiables into a set of values of linearly uncorrelated variables called Principal Components.

This technique is called Principal Component Analisys. The first component has the largest variance and each succeeding component

The first step consists to scan all images from the supplied dataset and insert them in an appropriate data structure. In this case, we have the matrix X containing all the images and the vector Y containing all the labels.

```python
from PIL import Image
import numpy as np
import os

X = [ ]
y = [ ]

for count, folder_name in enumerate(os.listdir("./PACS_homework/")):

    for file_name in os.listdir("./PACS_homework/"+ folder_name):
        img_data = np.asarray(Image.open("./PACS_homework/"+ folder_name +
                                         "/" + file_name))

        X.append(img_data.ravel())
        y.append(count)
```

STANDARDIZATION

In order to use the data, we have to standardize and normalize them. The following code standardizes the data, instead the normalization is done by the `transform` method.

```python
#Standardize
from sklearn import preprocessing

scaler = preprocessing.StandardScaler(copy=True, with_mean=True,
                                      with_std=True).fit(X)

X_scaled = scaler.transform(X)
```

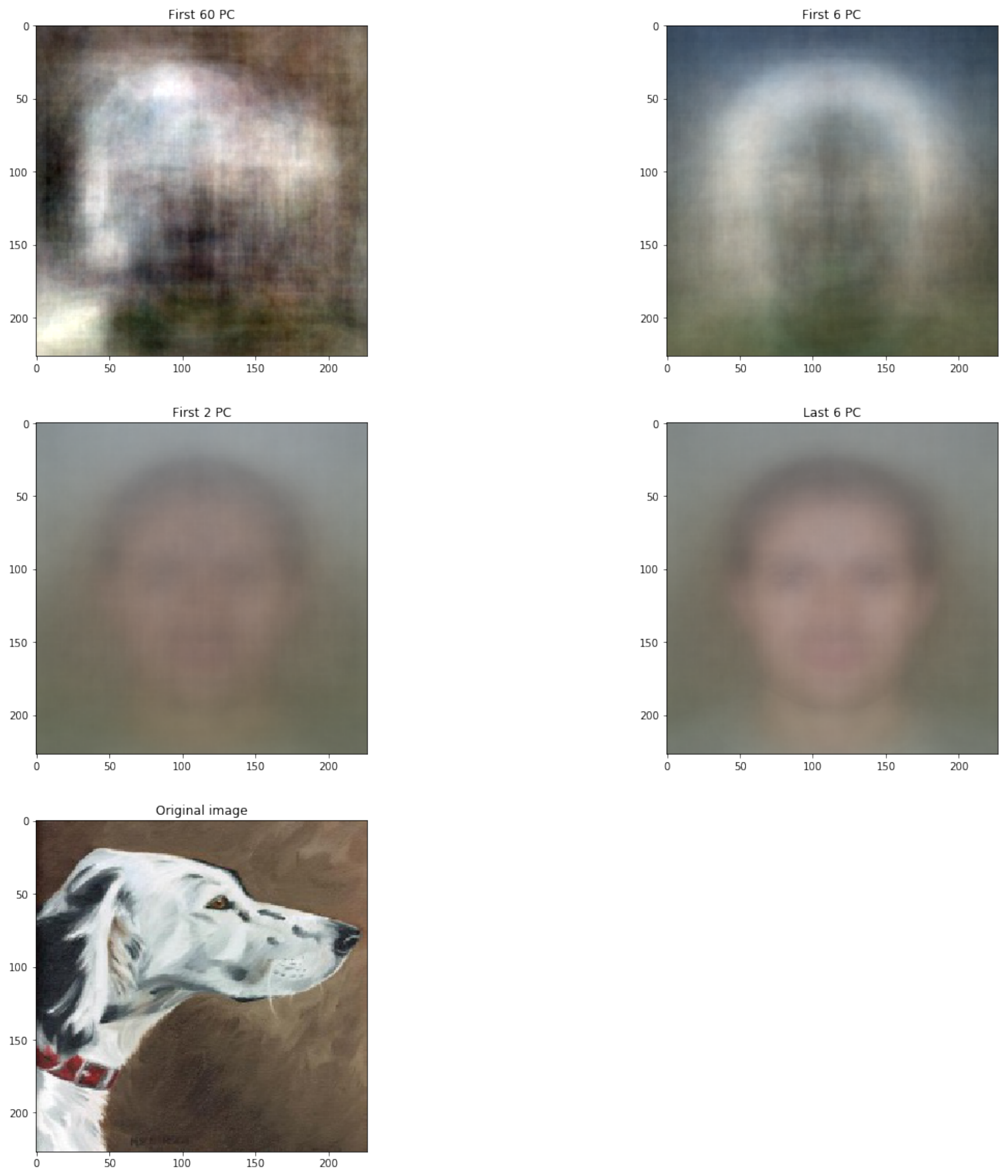
PART 1 - PRINCIPAL COMPONENT ANALISYS

Here, we use `PCA(n_components)` to extract the number of PC desired.

For the last six components, I took the function PCA with no parameters (that is keeping all the components) and then I manually extracted the last six overwriting them into the components_ attribute of the object returned by PCA.

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca_2 = PCA(2).fit(X_scaled)
X_t2 = pca_2.transform(X_scaled)
pca_6 = PCA(6).fit(X_scaled)
X_t6 = pca_6.transform(X_scaled)
pca_60 = PCA(60).fit(X_scaled)
X_t60 = pca_60.transform(X_scaled)
pca = PCA().fit(X_scaled)
pca_last6 = pca
#manually set last six components as components
pca_last6.components_ = pca_last6.components_[-6:,:]
X_tlast6 = pca_last6.transform(X_scaled)

#anti-transformation
X_inverse_2 = pca_2.inverse_transform(X_t2)
X_inverse_6 = pca_6.inverse_transform(X_t6)
X_inverse_60 = pca_60.inverse_transform(X_t60)
X_inverse_last6 = pca_last6.inverse_transform(X_tlast6)

#anti-standardization
X_descaled2 = scaler.inverse_transform(X_inverse_2)
X_descaled6 = scaler.inverse_transform(X_inverse_6)
X_descaled60 = scaler.inverse_transform(X_inverse_60)
X_descaled_last6 = scaler.inverse_transform(X_inverse_last6)

#printing images reconstructions
fig, ax = plt.subplots(3, 2, figsize=(20, 20))
ax[0, 0].imshow(X_descaled60[0].reshape(227,227,3).astype(int))
ax[0, 0].set_title("First 60 PC")
ax[0, 1].imshow(X_descaled6[0].reshape(227,227,3).astype(int))
ax[0, 1].set_title("First 6 PC")
ax[1, 0].imshow(X_descaled2[0].reshape(227,227,3).astype(int))
ax[1, 0].set_title("First 2 PC")
ax[1, 1].imshow(X_descaled_last6[0].reshape(227,227,3).astype(int))
ax[1, 1].set_title("Last 6 PC")
ax[2, 0].imshow(X[0].reshape(227,227,3).astype(int))
ax[2, 0].set_title("Original image")
ax[-1, -1].axis('off')
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for inte
```



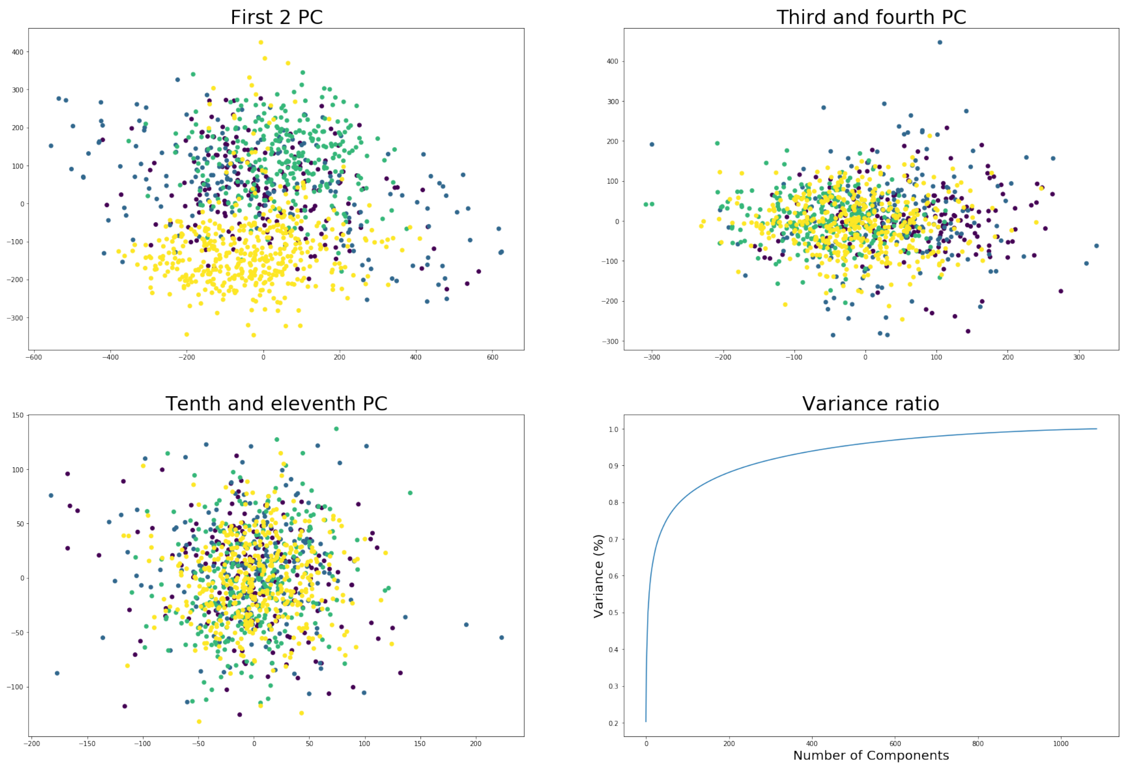
What we can deduce from these images is interesting. The images reconstructed using the first two PCs, the first six PCs and the last six PCs are very different from the starting image. This happens because as we have too little PCs we can not understand the image we are reconstructing, and so, due to the larger number of images representing people, it is more likely an human instead of another else.

If we got enough PCs, as we can see with the image with 60 PCs, then the image is more similar to the original one: in fact, if we try to reconstruct the image using all the possible components, we obtain that the reconstruced image is the same of the original one.

PART 2 - VISUALIZATION OF PROJECTED DATASET

```
fig, ax = plt.subplots(2, 2, figsize=(30, 20))

ax[0, 0].scatter(X_t2[:,0],X_t2[:,1],c=y)
ax[0, 0].set_title('First 2 PC', fontsize=30)
```

```
ax[0, 1].scatter(X_t6[:,2],X_t6[:,3],c=y)
ax[0, 1].set_title('Third and fourth PC', fontsize=30)
ax[1, 0].scatter(X_t60[:,9],X_t60[:,10],c=y)
ax[1, 0].set_title('Tenth and eleventh PC', fontsize=30)
ax[1, 1].plot(np.cumsum(pca.explained_variance_ratio_))
ax[1, 1].set_title('Variance ratio', fontsize=30)
ax[1, 1].set_xlabel('Number of Components', fontsize=20)
ax[1, 1].set_ylabel('Variance (%)', fontsize=20)

plt.show()
```



The first thing we can notice from these graphs is that as the PC "rank" increases, the distinction of the data is less visible. This happens because the eigenvalues that are multiplied by the covariance matrix (this multiplication gives us the eigenvectors) are sorted in descending order. As a matter of fact, the first eigenvalues (and so the first eigenvectors) will be the ones that contains most of the variance.

There are several different approaches for choosing the number of Principal Components needed to preserve data without much distortion, which are implemented depending on what type of informations is needed to deduce from the data. The ideal number of components needed is the one which contains the 99% of the variance of the data.

The last graph shows this concept.

PART 3 - CLASSIFICATION

In this part of the homework, we have to classify the dataset using a Naïve Bayes classifier.

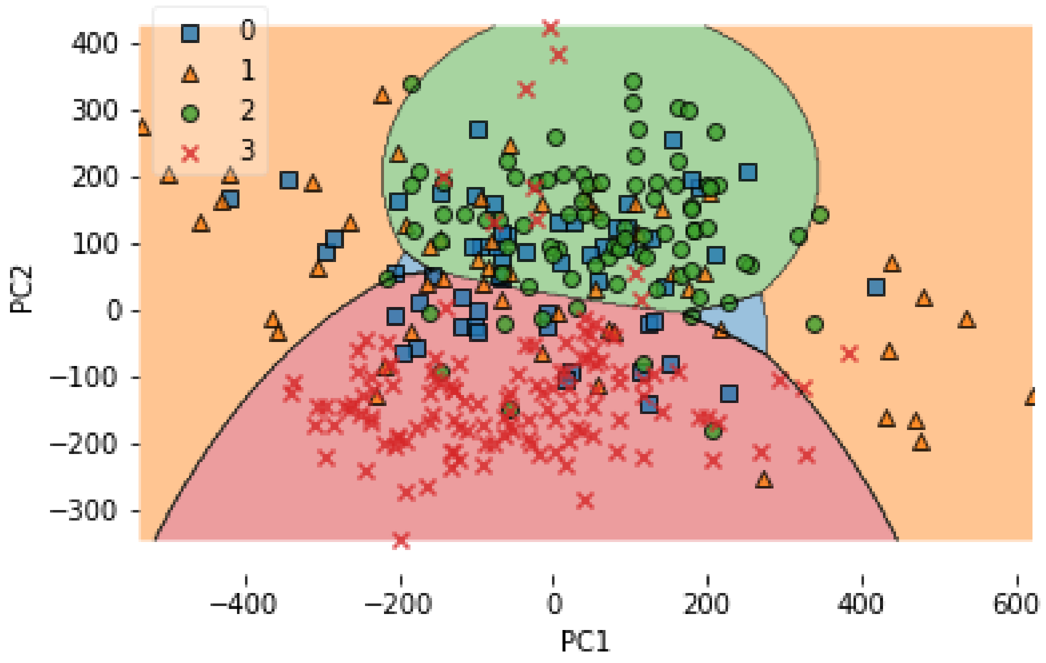
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
```

```python
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_decision_regions

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=5, stratify=y)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(classification_report(y_test, y_pred))
print("Confusion matrix:")
print()
print(confusion_matrix(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.55      0.70      0.62        57
          1       0.63      0.52      0.57        56
          2       0.78      0.82      0.80        84
          3       0.98      0.90      0.94       130

avg / total       0.79      0.78      0.78       327

Confusion matrix:

[[ 40   7   9   1]
 [ 20  29   7   0]
 [ 10   4  69   1]
 [  3   6   4 117]]
```

```python
X_train, X_test, y_train, y_test = train_test_split(X_t2, y, test_size=0.3,
                                                    random_state=5, stratify=y)
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(classification_report(y_test, y_pred))

plot_decision_regions(np.asarray(X_test), np.asarray(y_test), clf=gnb, legend=2)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
print("Confusion matrix:")
print()
print(confusion_matrix(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.33      0.02      0.03        57
          1       0.63      0.34      0.44        56
          2       0.53      0.85      0.65        84
          3       0.74      0.92      0.82       130

avg / total       0.60      0.64      0.58       327
```

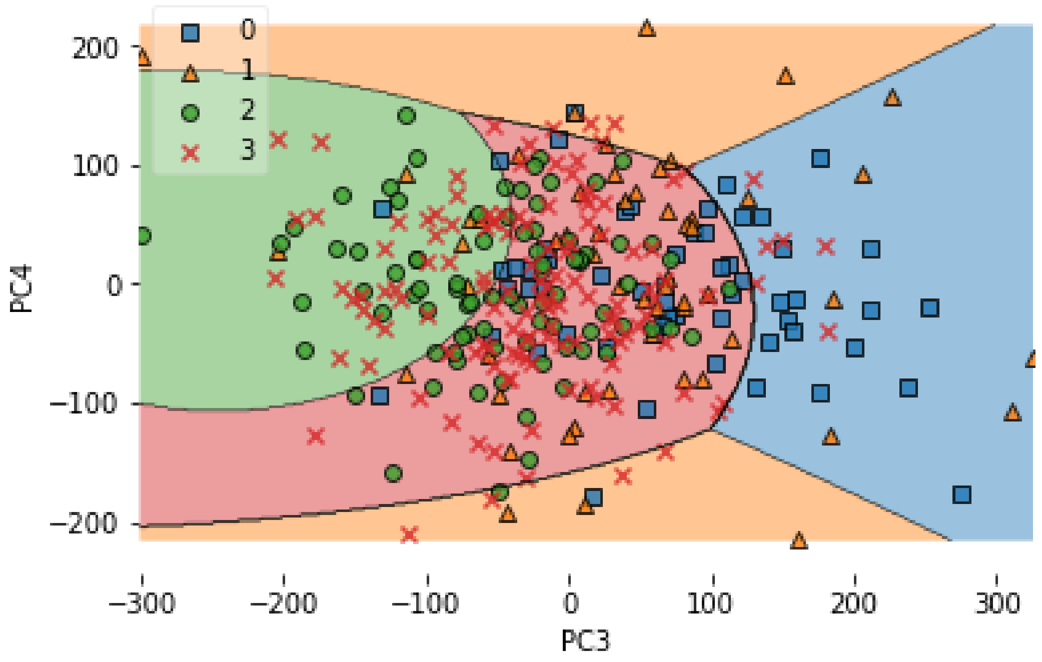


```
Confusion matrix:

[[  1   5  31  20]
 [  1  19  23  13]
 [  1   4  71   8]
 [  0   2   9 119]]
```

```
X_train, X_test, y_train, y_test = train_test_split(X_t6[:,[2,3]], y, test_size=0.3,
                                                    random_state=5, stratify=y)
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(classification_report(y_test, y_pred))

plot_decision_regions(np.asarray(X_test), np.asarray(y_test), clf=gnb, legend=2)
plt.xlabel('PC3')
plt.ylabel('PC4')
plt.show()
print("Confusion matrix:")
print()
print(confusion_matrix(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.58      0.32      0.41        57
          1       0.42      0.14      0.21        56
          2       0.40      0.33      0.36        84
          3       0.39      0.62      0.48       130
```

```
avg / total       0.43      0.41      0.39       327
```



```
Confusion matrix:

[[18  2  1 36]
 [ 7  8  6 35]
 [ 0  1 28 55]
 [ 6  8 35 81]]
```

The elements in the diagonal of the confusion matrix are the ones which are classified correctly. We clearly see that as we take "less important" components, more we have missclassifications, for example in the confusion matrix of the third and forth components, the correctly classified images are few, in fact 36 "dog" images are classified as "people".