Fondamenti di Machine Learning Project

Nicolò Squarzoni

a.a. 2021/2022

1) Introduction and project description

The chosen project is the number three between the provided list: "Given a set of human activities (e.g. walking, sitting, etc.), you should classify them".

It is provided a dataset of measurements collected from accelerometer and gyroscope of smartphones belonging to thirty volunteers involed in a specific experiment.
It was about collecting such data from the device while the volunteers were doing one of six different specific activities: WALKING, WALKING UPSTAIRS, WALKING DOWNSTAIRS, SITTING, STANDING and LAYING.
During the data processing 561 features have been identified from the measurements performed.
Each experiment has been labeled with the real activity performed and the dataset is divided in a training part and a test part.

The goal is to train a classifier and test it with the provided datasets.

2) Project realization

i) Strategy definition

The project has been coded with Python, mainly using Panda and Numpy libraries.
The first step has been the decision of performing PCA on the dataset to reduce the problem dimesionality and extract the most relevant features.
Then it has been decided to manually implement the classification algorithm provided in the slides to do face recognition, which seams to be a simplified version of KNN.
It would have been interesting to compare the results with KNN and LDA algorithms provided by Scikit-Learn library but time did not allow it.

ii) Implementation

The entire code is in a single Python file which features two main functions: PCA and recognition.

PCA, given training dataset, related lables and desired coverage applies PCA algorithm and projects the the dataset in a n_components dimensionality space, where n_components is calculated by the Ncomponents function that applying the formula (see below) seen on the slides chooses the n_components eigenvectors with the highest eigenvalues that guarantee the desired coverage.

$$\sigma_{\text{covered}} = \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{N} \lambda_j}$$

Once the dataset is in the new space PCA implements the first part of the classification algorithm which is the learning part and is related to the formula below:

$$\theta = \max\left\{\left\|\boldsymbol{\omega}^{(j)} - \boldsymbol{\omega}^{(k)}\right\|\right\} \text{per } j,k = 1,\ldots,M$$

where OMEGAJ and OMEGAK are supposed to be vectors representing the classes j and k between the list of classes corresponding to the lables.
As representing vector for each class it has been chosen to use the mean vector, so for each class it has been calculated the mean vector using all the data labeled with that specific class.
The result, TETA, is the maximum distance between the mean of the classes in the training dataset.
So at the end PCA function provides all the relevant calculated variables to perform the recognition phase.

This phase, for testing, is performed by the recognition function which needs the dataset to perform the recognition on, the lables and the relevant parameters related to PCA feature extraction and training phase provided by the PCA function.

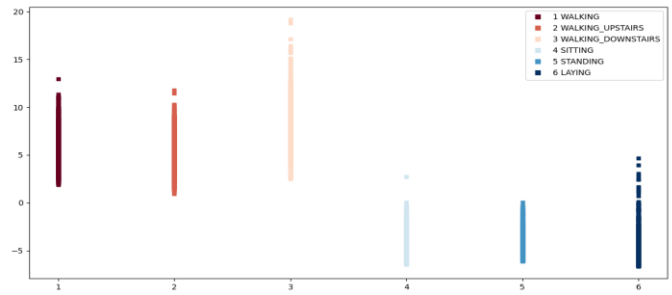
The algorithm performs the projection of the dataset in the reduced space previously calculated with PCA and in such space calculates the distance of each element of the dataset from the mean of each class storing the result in the list of lists EPSILON.

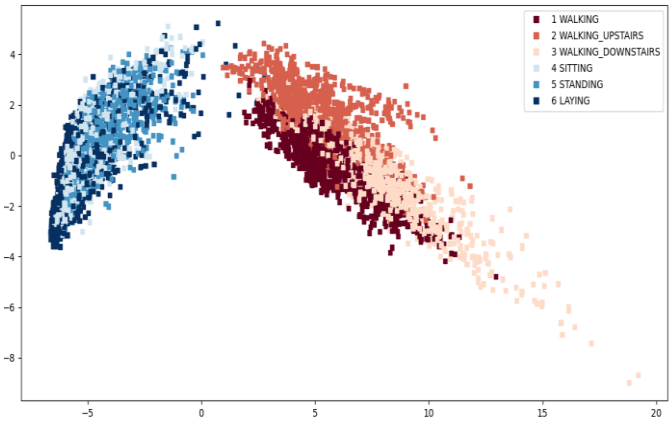
Then the function reverses the projection of the dataset back in the original space creating a second dataset e evaluates for each element the distance between its reversed version and the original version, both in the original space. The results are saved in a list of parameters called ZITA.

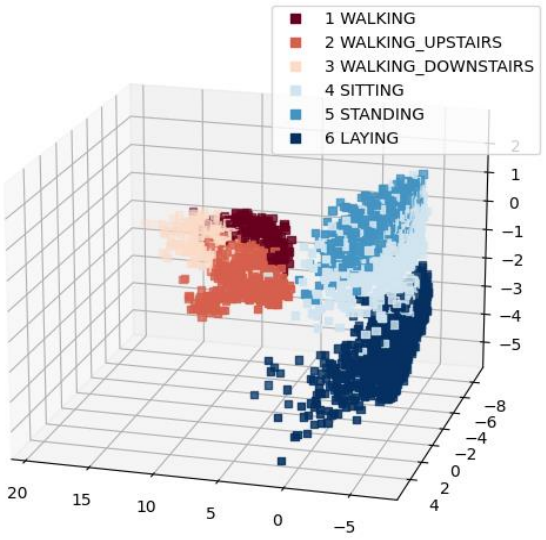
Then the last part of the algorithm evaluates each element assignign to it a specific class or other attributes depending on the related values of zita and epsilon.
Finally the results are saved in a list of lists where for each element it is provided the class or other attribute assigned, the real class from the label array and a boolean index stating if the classification was right or wrong.
The outputs are the afore mentioned list of list and an indicator of success of the algorithm on the test dataset.

Test

Some extra lines of code are used to load the datasets from the .txt files, call the functions, create the confusion matrx and print or plot the results.
The first step has been the run the PCA function and below are three images of the scatter plot of the dataset in three reduced spaces: 1d, 2d, 3d.







From these images it can be observed that the algorithm implemented is correct and the data representation is consistent: in fact it makes sense that data from wolking activities and data from still alctivities are respectively close to each other.
This also leads to assume that data from the accelerometer are effective in separating the classes while data from the gyroscope are not.

From these images it can also be observed that it might not be very easy to do classification because some classes are quite close to each other.

The performance of the classifier implemented are then reported below and different tests are shown chaging the dimensionality of the reduced space.

```
n dimensions:  1
Right guess:  47.16661011197829 %
Wrong guess:  52.86732270105192 %
Confusion matrix:
[[121 269 106   0   0   0]
 [ 86 346  39   0   0   0]
 [ 88 100 232   0   0   0]
 [  0   2   0  47 180 262]
 [  0   1   0  77 264 190]
 [  0   1   0  41 115 380]]
```

```
n dimensions:  2
Right guess:  54.02103834407872 %
Wrong guess:  46.01289446895148 %
Confusion matrix:
[[221 149 126   0   0   0]
 [ 60 383  28   0   0   0]
 [128  60 232   0   0   0]
 [  0   3   0  61 208 219]
 [  0   1   0  71 335 125]
 [  0   2   0  44 131 360]]
```

```
n dimensions:  3
Right guess:  70.37665422463523 %
Wrong guess:  29.657278588394973 %
Confusion matrix:
[[238 136 122   0   0   0]
 [ 54 386  31   0   0   0]
 [130  58 232   0   0   0]
 [  0   4   0 273 214   0]
 [  0   1   0 115 416   0]
 [  0   2   0   6   0 529]]
```

```
n dimensions:  5
Right guess:  77.02748557855446 %
Wrong guess:  23.00644723447574 %
Confusion matrix:
[[385  17  94   0   0   0]
 [ 24 417  30   0   0   0]
 [129  53 238   0   0   0]
 [  0   4   0 286 201   0]
 [  0   1   0 118 413   0]
 [  0   2   0   4   0 531]]
```

```
n dimensions:  9
Right guess:  81.64234815066169 %
Wrong guess:  18.39158466236851 %
Confusion matrix:
[[420  16  60   0   0   0]
 [ 21 422  28   0   0   0]
 [106  60 254   0   0   0]
 [  0   3   0 337 151   0]
 [  0   1   0  91 440   0]
 [  0   1   0   3   0 533]]
```

```
n dimensions:  34
Right guess:  83.6104513064133 %
Wrong guess:  16.4234815066169 %
Confusion matrix:
[[434  12  50   0   0   0]
 [ 23 426  22   0   0   0]
 [101  59 260   0   0   0]
 [  0   3   0 350 138   0]
 [  0   1   0  73 458   0]
 [  0   1   0   0   0 536]]
```

First of all it can be observed that the implemented algorithm works quite well and beyond the expectations.
Also it can be observed that increasing the dimensionality and thus increasing the coverage the algorithm becomes more accurate but tends to saturate raising over 10 dimensions, while the best improvement is from 2 to 3 dimensions.

The last aspect to oberseve from the confusion matrix is that when the algorithm fails it makes the mistakes between the walking classes or between the still classes, which matches with the scatter plots observed in the images above.