Introduction to Machine Learning

Work 2

Principal Component Analysis Exercise

Authors: Aleix Toda Mas, Alvaro Romero Diaz, Josep Famadas

November 18, 2017

Contents

Chapter 1

Description of the work

Multivariate Analysis often starts out with data involving a substantial number of correlated variables. Principal component analysis (PCA) is a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. This procedure is called dimensionality reduction or data compression. It should be noted that there is no guarantee that the obtained dimensions are interpretable.

PCA is one of the most popular techniques for processing, compressing and visualising data, although its effectiveness is limited by its global linearity. While nonlinear variants of PCA have been proposed, an alternative paradigm is to capture data complexity by a combination of local linear PCA projections. However, conventional PCA does not correspond to a probability density, and so there is no unique way to combine PCA models.

The aim of the exercise is to analyse with Principal Component Analysis (PCA) algorithms several data sets from the UCI repository. In first instance, the PCA extraction method is done step by step, and at the end the same procedure is done by the PCA algorithm included in the sklearn package.

This procedure will be done iteratively under 3 different UCI datasets.

Chapter 2

Implemented additional functions

The back bone of our code resides in the main.py file, from where all functions are called and also all results are presented, but some functions have been implemented in order to provide a cleaner code and also to avoid rewriting blocks of similar code repeatedly. This functions are located in separate .py files, presented following.

2.1 datasets.py

This file only manages the selection of the desired dataset. It contains some UCI datasets and it's absolute path to them, contained in an array. Passing a number from 0 to 14 as an input, it returns the selected complete dataset path.

2.2 parserPCA.py

The first part of any PCA method consists of parsing the input data in order to have it in a format in which the PCA algorithm is able to work with.

In the case of this project, the parser will load the data from the arff file determined by the input variable path using the function loadarff, contained in the library scipy. When a database is loaded it is a list of lists where each row corresponds to an instance and each column corresponds to one of the attributes, except for the final column where there are specified the real class of each instance.

The functionality of the parsing function designed in this project is to split the initial data into a numpy matrix containing only the instances and its attributes and another numpy array which contains the classes. Once the split has been done, it is not returned directly, there are some pre-process made on the data, iterating the attributes (columns):

- If the attribute is numerical, the ?nan?s are substituted by the average of the column. If the most common value in the attribute is ?nan?, the column is erased because there

is not enough data to take it into account for the clustering. After that, the column is normalized in a way that the minimum value is 0 and the maximum is 1, so the maximum distance is 1.

- If the attribute is nominal, the ???s are substituted by the most common value of the column. If the most common value in the attribute is ???, the column is erased because there is not enough data to take it into account for the clustering.
- PCA can't deal with categorical data so, here each unique element of the categorical features is substituted by a integer numerical value, from 0 to the length of the unique elements in the feature.
- Once previous preprocessing is done, the numerical data is normalized, fixing 1.0 as the greater number and 0.0 as the minimum.

Finally, the parser function returns the preprocessed data and the classes in numpy matrix and numpy array formats correspondingly as previously said.

2.3 plotFeatures.py

In this file there are 3 different functions, mainly related to the plotting of data.

1. The first is the **chooseXfeatures** function, that uses the data instances, the labels and the number of features that we would like to have after performing PCA extraction. This function use the sklearn functions **RFE** and **LogisticRegression** from the *feature_selection* and *linear_model* packages respectively, to select the most important features from the data.

   Recursive feature elimination (RFE): Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a *coef_* attribute or through a *feature_importances_* attribute. Then, the least important features are pruned from current set of features.That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

   Logistic regression: Despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as *logit* regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

2. The second function is has been named **labelsSetting** and it is used to label the axis plots properly given the name of the different axis.

3. The last and a little bit more complex is the **plotXfeatures**, it's inputs are the instances data, an array with the indices of the selected features, a parameter called multiple, a string containing the title of the plot and the desired axis labels.

   The behaviour of this function begins detecting if the desired plot is 2D or 3D and depending on the value of the parameter multiple, different actions could be done:

   - If multiple is 0, a simple plot is done.
   - If multiple is 1, the data input has to be a list of different list of data instances, so the idea is to overlap different plots in one only figure.
   - If multiple is 2, the data input has to be a list of different list of data instances, and each list of instances is plotted in a different subplot of the same figure.

Chapter 3

Algorithm to perform PCA step by step

The work description has been used as the skeleton of our algorithm:

Step 1. Read the .arff file and take the whole data set consisting of d-dimensional samples ignoring the class labels. Save the information in a matrix

Here the created function getDatasetPath is used in order to provide the corresponding dataset path and pass it to the parser function, that as previously explained, clean the dataset, normalize it and load it to a numpy matrix, retrieving the data instances on the one hand and a numpy array containing the classes (labels) in the other hand.

Step 2. Plot the original data set (choose two or three of its features to visualize it).

Here appears the numberOfFeatures variable, that define the number of dimensions that we would like to retain after performing PCA. Then, the chooseXfeatures is used to return the indices of the features that present less correlation with the others. Next some figures plotting the different used datasets taking into account only the 2 or 3 features (depending on the case) are presented.

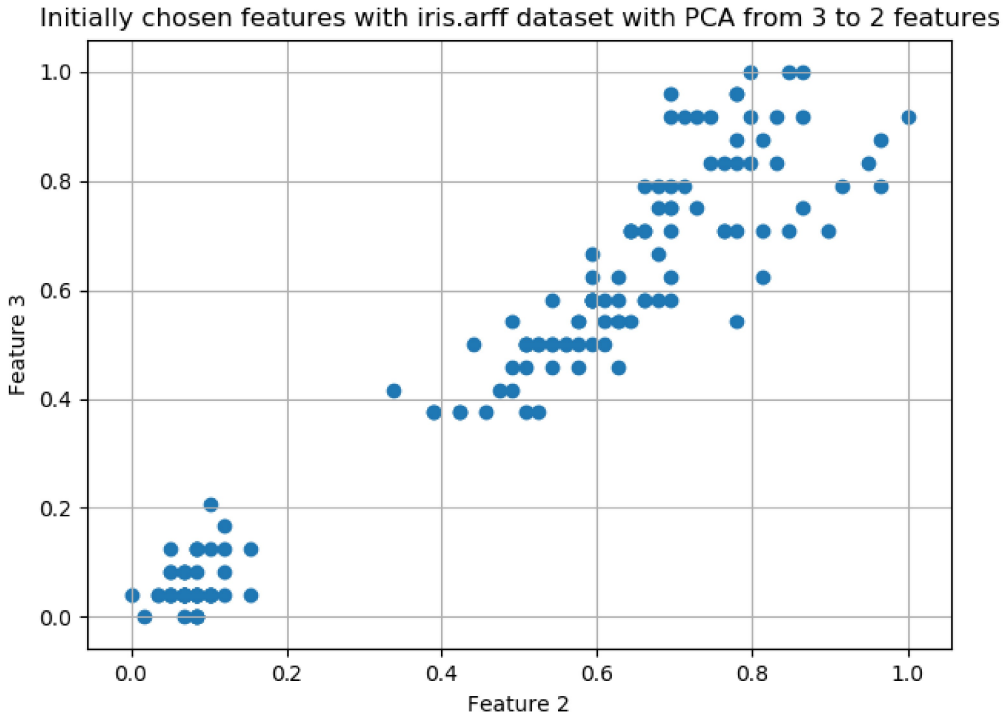

Figure 3.1: Iris data only taking into account the 2 most uncorrelated dimensions

Step 3. Compute the d-dimensional mean vector (i.e., the means of every dimension of the whole data set).

In step 3 the mean of each feature is extracted and then it is subtracted element by element of the data matrix.

Step 4. Compute the covariance matrix of the whole data set. Show this information.

Here the covariance matrix is obtained with the *numpy.cov()* function, given the centred data. The obtained covariance matrix is printed in the console.

$$A = \begin{pmatrix} 0,0529 & -0,0045 & 0,0600 & 0,0598 \\ -0,0045 & 0,0326 & -0,0227 & -0,0205 \\ 0,0600 & -0,0227 & 0,0894 & 0,0916 \\ 0,0598 & -0,0205 & 0,0916 & 0,1011 \end{pmatrix}$$

Covariance matrix of iris dataset A

Step 5. Calculate eigenvectors (e1, e2, ..., ed) and their corresponding eigenvalues of the covariance matrix. Write them in console.

In this step eigenvectors are calculated with the numpy.linalg.eig() function, that retrieves both the eigenvalues and the eigenvectors, ordered in some random way but linked by the index between each pair eigenvalue-eigenvector.

This values are also printed in the console. Following the eigenvalues and eigenvectors obtained with the iris dataset are presented.

$$Eigenvalues = \begin{bmatrix} 0,2323 \\ 0,0323 \\ 0,0096 \\ 0,0017 \end{bmatrix}$$

$$Eigenvectors = \begin{bmatrix} 0,4253 & -0,1461 & 0,6161 & 0,6467 \\ -0,4210 & -0,9047 & 0,0643 & 0,0112 \\ -0,7143 & 0,3351 & -0,0683 & 0,6105 \\ 0,3628 & -0,2188 & -0,7821 & 0,4571 \end{bmatrix}$$

Step 6. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a new d x k dimensional matrix (where every column represents an eigenvector). Write the sorted eigenvectors and eigenvalues in console.

As indicated the eigenvalues are ordered in descending value and then the first p eigenvalues and eigenvectors, where p is the number of selected features after PCA extraction settled previously, are selected.

Next, the eigenvector with more weight for the iris dataset is presented.

$$\lambda^1 = \begin{bmatrix} 0,2323 \end{bmatrix} \qquad Eigenvector^1 = \begin{bmatrix} 0,4253 & -0,4210 & -0,7143 & 0,3628 \end{bmatrix}$$

The other chosen eigenvectors are selected with the same method, with each eigenvalue associated with one column of the eigenvector column.

It is important to note that even if the most important vectors are selected, all of them are used to obtain the transformed data in the next step.

Step 7. Derive the new data set. Use this d x k eigenvector matrix to transform the samples onto the new subspace.

The transformed data is obtained multiplying the eigenvectors matrix (named *rowFeaturesVector* in order to hold the same name as in the class theory) with the transpose of the centred data.

$$TransformedData = RowFeatureVector\ x\ RowDataAdjust \tag{3.1}$$

The transformed data is the centred data with a rotation applied. That rotation place the data eigenvector into the x axis.

As stated in the previous step, transformed data has been calculated with all the eigenvectors matrix. That has been done to maintain the possibility of reconstructing the transformed data into the original data using the inverse way followed to reach this point.

Step 8. Plot the new subspace (choose the largest eigenvectors to plot the matrix).

In this step the transformed data is plotted with the centred data in order to compare the performed rotation. Also 2 different transformed data are plotted, one with the data reduced to 2 features and another with the data reduced to only one feature.

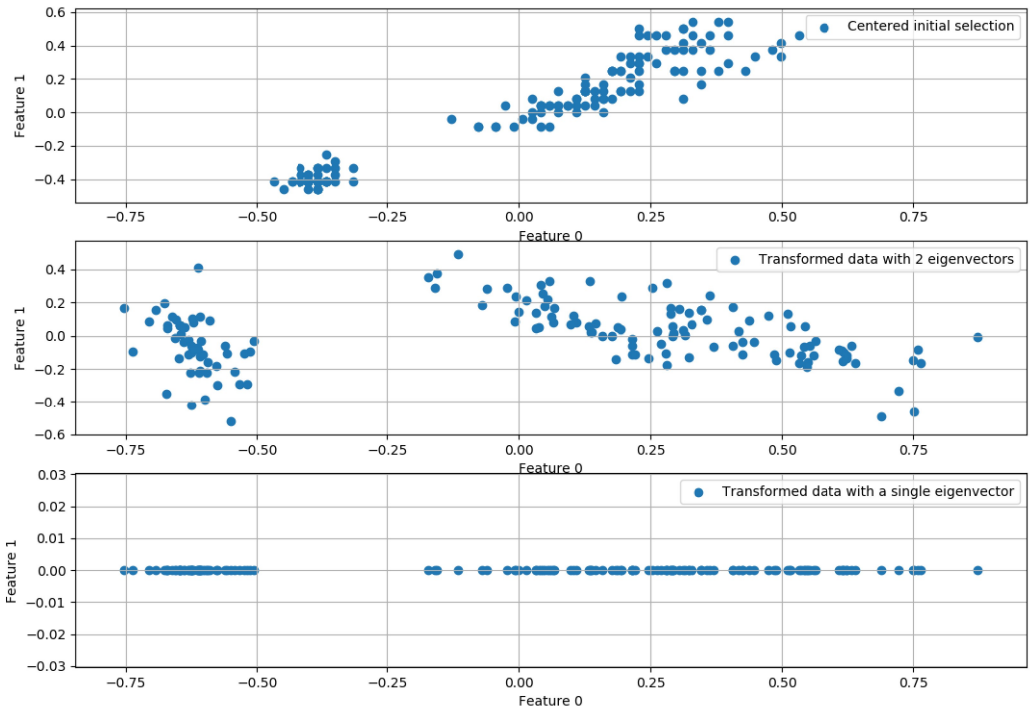

Figure 3.2: Above subplot: Original iris data only taking into account the 2 most uncorrelated dimensions. Middle subplot: Transformed data with 2 eigenvectors. Below subplot: Transformed data with a single vector

Step 9. Reconstruct the data set back to the original one. Additionally, plot the data set.

Finally, in the step 9 the inverse way is followed in order to recover the original data. This step is possible because we have retained all eigenvectors. If any eigenvector had been lost, this dimension would be lost and it will be impossible to recover. It also has to be said that if the lost eigenvectors had a little weight ratio, the lost information is not critical.

The reconstructed data adjusted (centred data) is obtained by the equation:

$$RowDataAdjust = RowFeatureVector^T \; x \; TransformedData \tag{3.2}$$

And then, using the equation:

$$RowDataOriginal = RowDataAdjust + OriginalMean \tag{3.3}$$

To add the previously subtracted mean, the original data is obtained again.

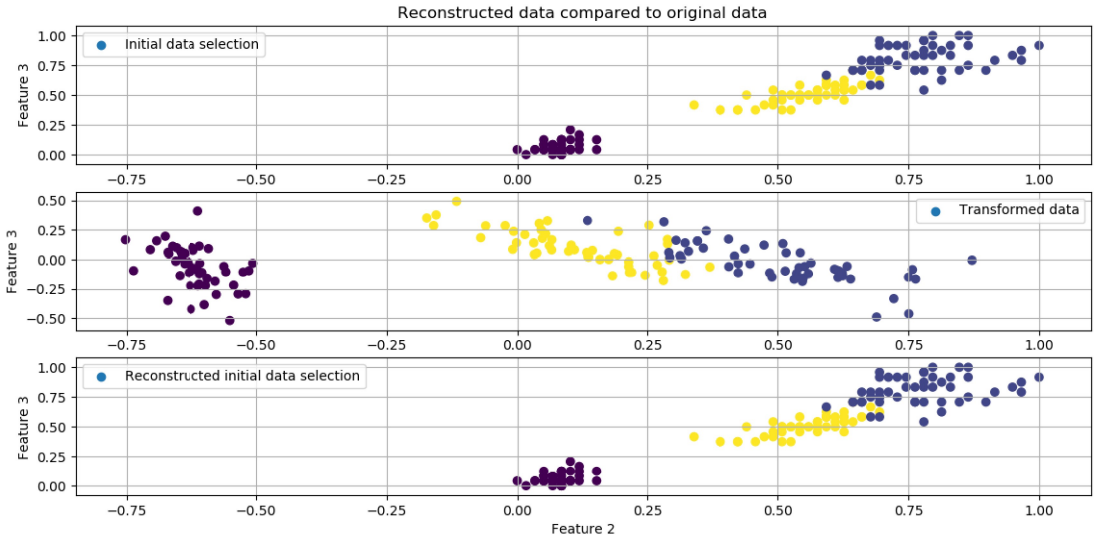

Figure 3.3: From above to below: First subplot: Original iris data only taking into account the 2 most uncorrelated dimensions. Second subplot: Centred data. Third subplot: Transformed data with 2 eigenvectors. Fourth subplot: Reconstructed original data.

In the 3.4 it could be seen that the reconstructed data present the same distribution as the original data.

3.1 To how many components should we have to reduce our dataset?

In order to ask this question we have used the score function of the sklearn package in order to evaluate our PCA dimension reduction for different number of final dimensions reduction, with the obtained results showed in the following figure for the case of the iris dataset.

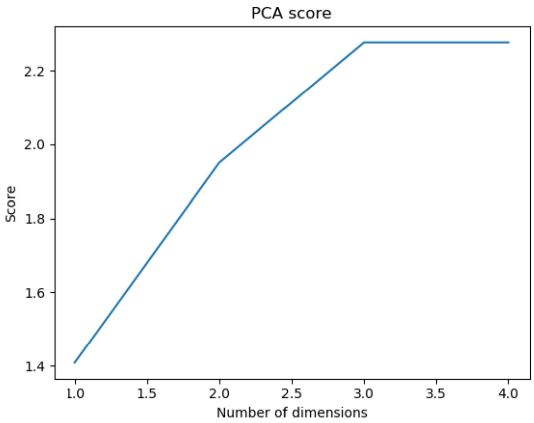

Figure 3.4: Scores for different final dimensions after performing PCA analysis.

Chapter 4

PCA with *sklearn.decomposition* package

Now, the same procedure done before is done automatically by the sklearn package. Sklearn provides a lot of tunable parameters with the aim to obtain the most reliable principal components data. In this case, we have only settled the number of components that we would like to achieve the reduction.

Some relevant parameters as covariance matrix, selected eigenvectors, variance ratio, singular values and precision are printed to the console.

The obtained results are approximately the same but a very important difference, with this algorithm it is not possible to reconstruct the original data due that when PCA is performed, the less important eigenvectors are discarded when obtaining the eigenvector matrix and then the obtained reconstructed data has been lost some dimensions.

The sklearn package provides a useful tool ($score(X[, y])$) that returns the average log-likelihood of all samples. This parameter could be used to iterate over some final dimension values and then compare between the obtained values. As higher the score is for a particular final dimensions size, the better is to choose that number of dimensions.

In figure 4.1, ahieved results could be seen.

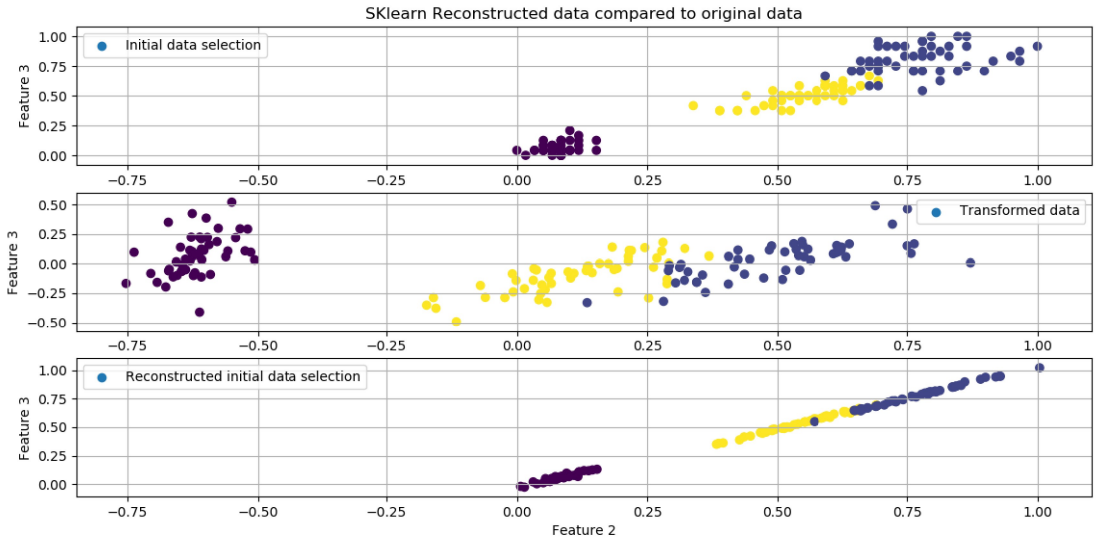

Figure 4.1: From above to below: First subplot: Original iris data only taking into account the 2 most uncorrelated dimensions. Second subplot: Transformed data with 2 eigenvectors. Third subplot: Reconstructed original data.

Chapter 5

Some examples of PCA with other datasets

In this section the explained step by step algorithm is performed in different datasets.

5.1 Adult dataset

Adult dataset has either numerical as categorical features, also it has such a large amount of different features as 14. When the 3 features with less correlation with others are plotted in a 3 dimension plot, the next figure is obtained:

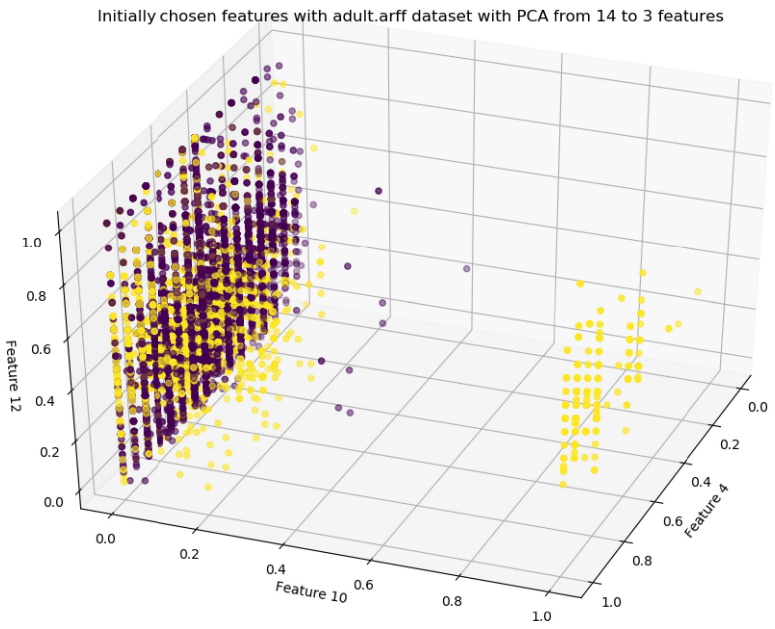

Figure 5.1: Original adult data only taking into account the 3 most uncorrelated dimensions.

Initially it seems that it is possible to easily classify the most of the data. The ratio of the obtained eigenvalues is the following:

$$\lambda(ratio)_{(1\,to\,7)} = \begin{bmatrix} 0,369 & 0,127 & 0,105 & 0,092 & 0,071 & 0,060 & 0,007 \end{bmatrix}$$

$$\lambda(ratio)_{(8\,to\,14)} = \begin{bmatrix} 0,007 & 0,011 & 0,019 & 0,038 & 0,029 & 0,033 & 0,032 \end{bmatrix}$$

As could be seen, the first eigenvalue is for far the one that give us most information about the different clusters of data, but then, from the 2nd to the 6th have approximately the same ratio. If we try to do the PCA transform with only the 3 first eigenvectors, we obtain the following figure:

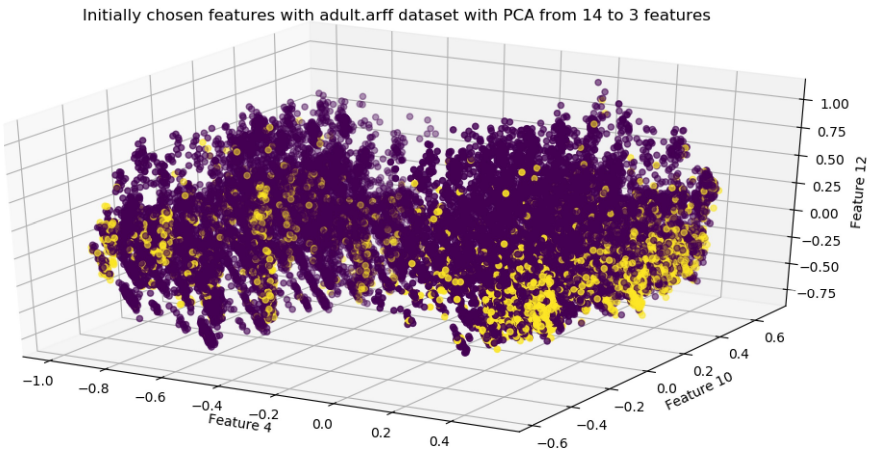

Figure 5.2: Transformed adult data and recovery only taking into account the 3 most uncorrelated dimensions.

In figure 5.2 it is clearly seen that in this case a PCA don't perform properly if we only take into account the most uncorrelated 3 features.

In figure 5.3 it could be noted that the achieved score with less than 6 features is quite poor, and over 11 features we will achieve an accurate result, so we only could discard 3 features in this case.

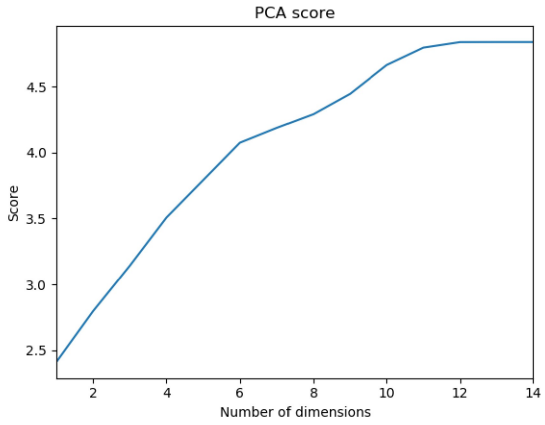

Figure 5.3: Adult data scores.

5.2 Breast Wisconsin dataset

Breast Wisconsin dataset has only numerical features, also it has such a large amount of different features as 9. When the 2 features with less correlation with others are plotted in a 2 dimension plot, the next figure is obtained:

imgs/originalData_breast.png

Figure 5.4: Original breast data only taking into account the 2 most uncorrelated dimensions.

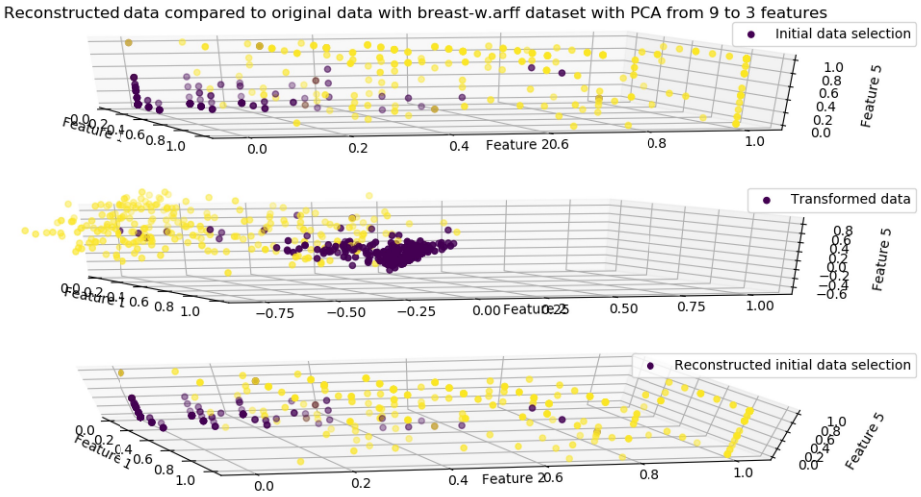

Figure 5.5: Original breast data only taking into account the 3 most uncorrelated dimensions.

Initially it seems that it is possible to easily classify the most of the data. The ratio of the obtained eigenvalues is the following:

$$\lambda(ratio) = \left[\begin{array}{cccccccc} 0,689 & 0,073 & 0,061 & 0,011 & 0,044 & 0,039 & 0,035 & 0,023 & 0,025 \end{array} \right]$$

As could be seen, the first eigenvalue is for far the one that give us most information about the different clusters of data, the second and third give some information and the rest are dispensable. If we try to do the PCA transform with only the 2 first eigenvectors, we obtain the following figure:

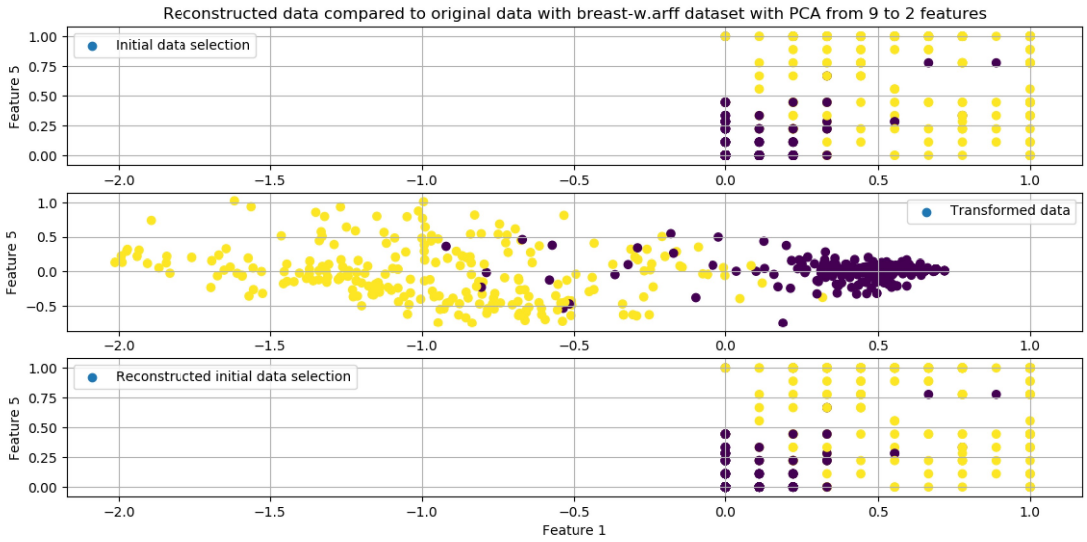

Figure 5.6: Breast data transformation and recovery only taking into account the 2 most uncorrelated dimensions.

In figure 5.6 it is clearly seen that in this case a PCA perform with a high accuracy and then it could be very useful in order to save time grouping the data or also provides a good way to visualize the differences between the data.

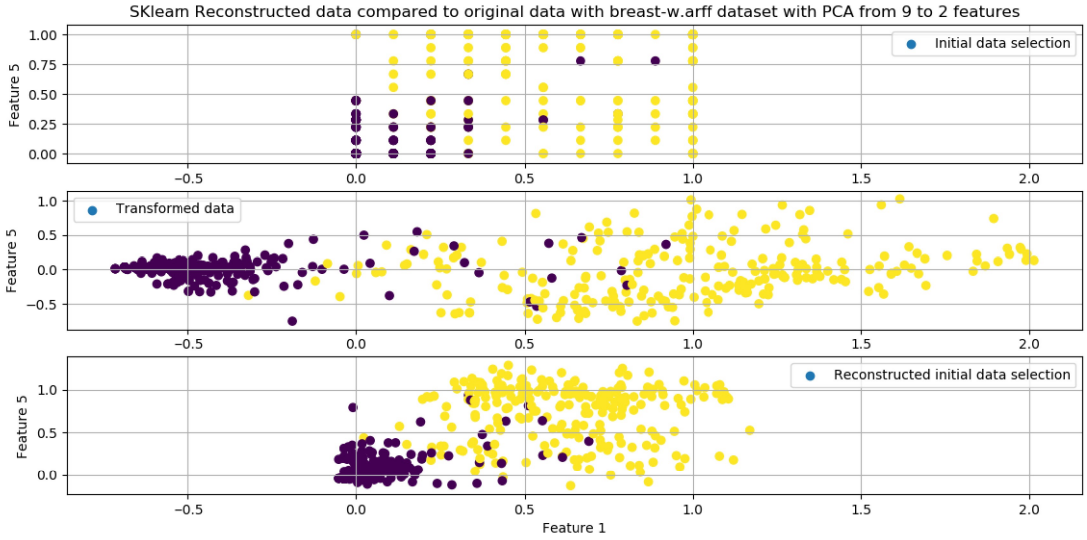

Figure 5.7: Breast data transformation and recovery with sklearn only taking into account the 2 most uncorrelated dimensions.

In figure 5.8 it could be noted that the achieved score with less than 3 features has a high score but could be too less so, a good compromise solution could be to choose 3 as the number of remaining features after performing PCA.

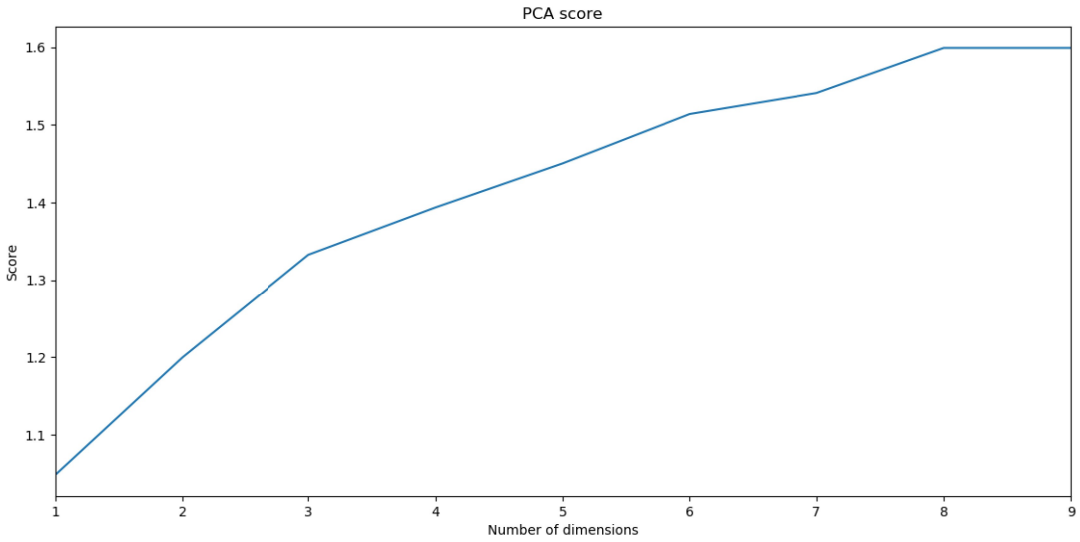

Figure 5.8: Score breast data.

5.3 Satimage dataset

Satimage dataset has only numerical features, also it has such a large amount of different features as 36. When the 2 features with less correlation with others are plotted in a 2 dimension plot, the next figure is obtained:

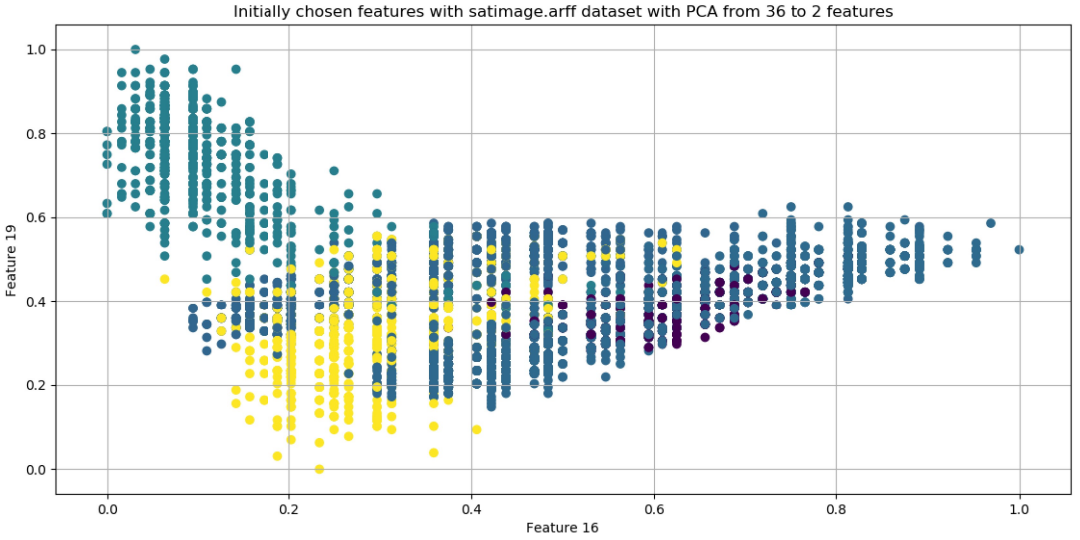

Figure 5.9: Original satimage data only taking into account the 2 most uncorrelated dimensions.

At first sight it is clear that only with 2 features the achieved accuracy will be very low. The ratio of the obtained eigenvalues is the following:

$$\lambda(ratio)_{(1\,to\,9)} = \left[\begin{array}{ccccccccc} 5,40E-01 & 3,06E-01 & 5,28E-02 & 2,53E-02 & 1,82E-02 & 1,54E-02 & 9,17E-03 & 5,12E-03 & 3,58E-03 \end{array}\right]$$

$$\lambda(ratio)_{(10\,to\,18)} = \left[\begin{array}{ccccccccc} 3,42E-03 & 2,19E-03 & 1,75E-03 & 1,59E-03 & 1,67E-03 & 1,25E-03 & 1,16E-03 & 1,14E-03 & 1,07E-03 \end{array}\right]$$

$$\lambda(ratio)_{(19\,to\,27)} = \left[\begin{array}{ccccccccc} 8,14E-04 & 7,49E-04 & 7,26E-04 & 6,98E-04 & 6,23E-04 & 2,09E-04 & 5,66E-04 & 5,46E-04 & 2,55E-04 \end{array}\right]$$

$$\lambda(ratio)_{(28\,to\,36)} = \left[\begin{array}{ccccccccc} 4,82E-04 & 2,84E-04 & 3,05E-04 & 3,34E-04 & 4,21E-04 & 4,14E-04 & 3,85E-04 & 3,69E-04 & 3,72E-04 \end{array}\right]$$

The transformed data is the following:

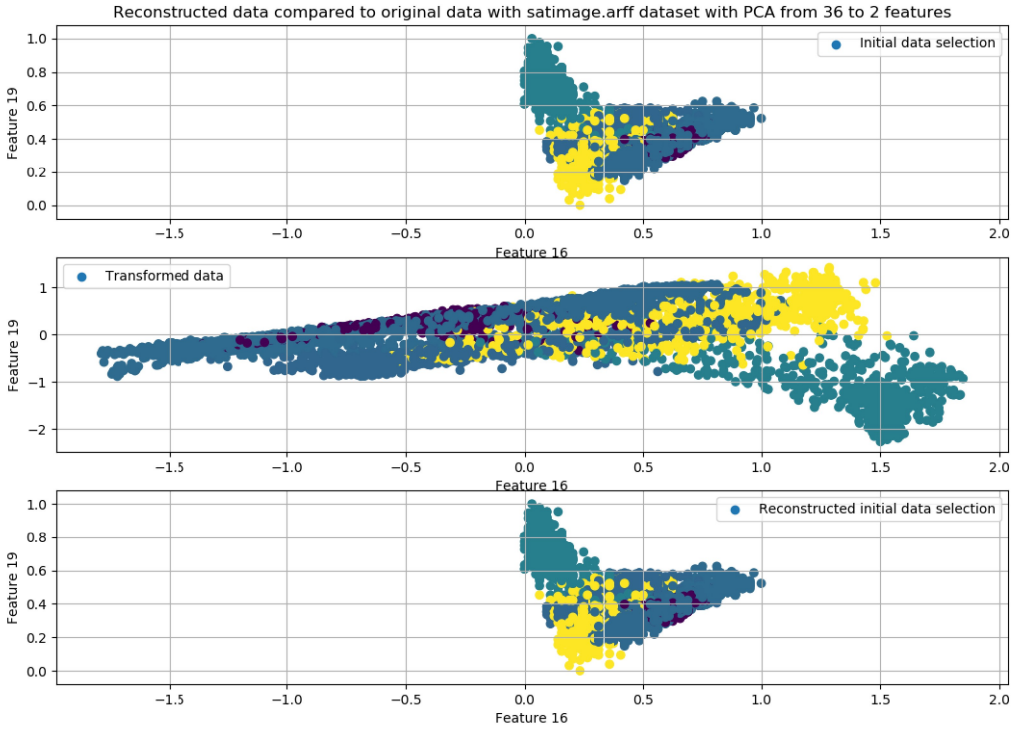

Figure 5.10: Transformed satimage data and recovery only taking into account the 2 most uncorrelated dimensions.

And the presented score is the next:

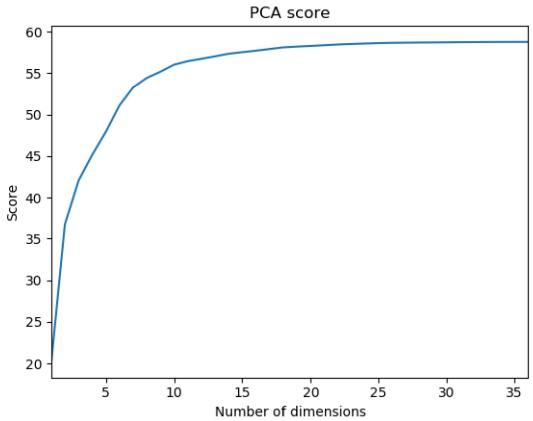

Figure 5.11: Satimage data scores.

Chapter 6

Instructions to execute the program

In order to execute the program, simple instructions have to been followed:

1. Set the **choosenDataset** variable to the desired dataset index. In the 6.2, the principally used datasets could be seen, and in **??**, detailed information about each of them could be seen.
2. Run the program, it automatically will show all graphics and will print the most important information in the console.

	Features			
	Num	**Cat**	**Clusters**	**Dev**
adult.arff dataset	6	8	2	26,07%
kropt.arff dataset	0	6	18	5,21%
bal.arff dataset	4	0	3	18,03%
waveform.arff dataset	40	0	3	0,36%
pen-based.arff dataset	16	0	1	0,40%
breast-w.arff dataset	9	0	2	20,28%
satimage.arff dataset	36	0	6	6,19%
iris.arff dataset	4	0	3	-

Table 6.1: Used datasets

Datasets		
Index	**Name**	**Clusters**
0	iris.arff	3
1	glass.arff	2
2	grid.arff	2
3	adult.arff	2
4	hepatitis.arff	2
5	autos.arff	6
6	kropt.arff	18
7	waveform.arff	3
8	pen-based.arff	10
9	satimage.arff	6
10	pima_diabetes.arff	2
11	satimage.arff	6
12	bal.arff	3
13	breast-w.arff	2
14	soybean.arff	19

Table 6.2: Available datasets