PCA

Federico Gianno

***Abstract*—Application of PCA applied on images. It shows what happens if different Principal Components (PC) are chosen as basis for images representation and classification. Then, will be chosen and applied a classifier in order to classify the images under different PC re-projection.**

I. Introduction

Principal component analysis (PCA) is a **statistical procedure** that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called *principal components* [1].

If there are *n* observations with *p* variables, then the number of distinct principal components is

$$min(n-1, p)$$

This transformation is defined in such a way that the first principal component has the largest possible variance.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models.

II. Theory

Given data points into d-dimensional space, project them into a lower dimensional space while preserving as much information as possible. In particular, it's needed to choose projection that minimizes squares error in reconstructing original data.

Principal Components are points in the direction of the largest variance. Each subsequent principal components:

- is orthogonal to the previous ones
- points in the direction of the largest variance of the residual subspace

III. Procedures

This section briefly describes a python code implementation reporting several **snippet code** with the corresponding explanations and motivations.

1) Requirements
   - Pillow (imported as Image)
   - numpy (imported as np)
   - scikit-learn
   - matplotlib (imported as plt)
2) Data preparation
   The given data-set is composed by different kind of photos:
   - Dog
   - Guitar
   - House
   - Person

   First of all I need to read the data-set and create a matrix *x* in which each row is an 1-D array associated to an image.

   ```
   img3d = np.asarray(Image.open(path))
   ```

   The function will return a 3-D array of the corresponding image. I need to scale back the 3-D array into 1-D array.

   ```
   img1d = img3d.ravel()
   ```

3) Normalization
   PCA is effected by scale so I need to scale the features in the data before applying PCA.
   The normalization consists of mean centering (zero mean) and normalizing each variables variance to make it equal to one (standard deviation equals to one).

   ```
   x = (x-np.mean(x))/np.std(x)
   ```

4) PC extraction
   At this point I need to extract the *n* principal components from matrix *x*, fit the model with matrix and apply the dimensionality reduction to the matrix.

   ```
   # For example 2-PC
   pca2 = PCA(n_components=2)
   x_t_2 = pca2.fit_transform(x)
   ```

   The original data which is *n-dimensional* it's converted into *n_components-dimensions*.
   It can be useful to print the **explained variance**.

   ```
   pca2.explained_variance_ratio_.cumsum()
   ```

   The explained variance tells me how much information (variance) can be attributed to each of the principal components: it is the ratio between the variance of that principal component and the total variance.
5) PC visualization and Image reconstruction
   Now I am able to visualize the PC. I can plot the whole data-set or a single photo *p* from the data-set:

   ```
   plt.scatter(x[p,0], x[p,1], c=color)
   plt.show()
   ```

   And after having rebuilt the images I can show it too

   ```
   xr2 = pca2.inverse_transform(x_t_2)
   # De-Standardizing matrix
   xr2 = (xr2*std)+mean
   # From 1-D to 3-D
   img2 = np.reshape(xr2[p], (227, 227, 3))
   plt.imshow(img2, interpolation='nearest')
   plt.show()
   ```

IV. Image re-projection

Each folder contains a lot of images, then the data-set is huge and the PCA approach will be good: (in the PCA) the more are the data the more accurate the final result will be.

After the first main steps described in III, I chose a random image from the whole data-set and i plotted it considering the first 60-PC, 6-PC, 2-PC and the last 6-PC.

For the *last* n-PC the approach is the same of the *first* n-PC with a little difference: instead of sorting the [eigenvalues, eigenvectors] tuples from the high to the low, in order to have the largest possible variance, we have to sort it from the low to the high. From the point of view of the code implementation you need to do a manual PC computation.

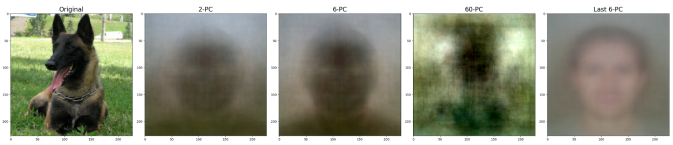

Fig. 1. *Whole data-set*. Respectively from the left to the right: original image, the first 2-PC, 6-PC, 60-PC and last 6-PC.

As you can see the 60-PC (3rd image) image is quite similar to the original one. In fact the more are the PC the more similar to the original image will be to the re-projected one.

On the other hand, quite strange are the 2-PC and the 6-PC: the original photo is a dog but the 2-PC and 6-PC looks like a human face. This is due to the data-set: the matrix contain the whole data-set that is composed by a lot of different images (dog, guitar, house and person) and not only by dogs. So, when the PC are computed, are brought also PC of different classes: in this case the PC of the human face have more impact than the PC of the dog and so the first PC looks like a human face instead of a silhouette of a dog.

The last 6-PC, are the PC with the lowest possible variance, so the features that does not allow to distinguish in a good way the images from each others. In this case it looks like a human face and it is due to the data-set that, as I said before, is composed by several kind of images.

At this point I chose to read only one subject of images. The obtained results is showed in Fig. 2.

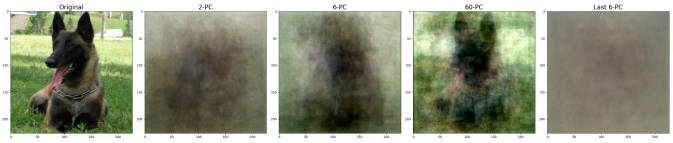

Fig. 2. *Dogs images*. Respectively from the left to the right: original image, the first 2-PC, 6-PC, 60-PC and last 6-PC.

The result, as you can see, it's different. Now, the 2-PC and 6-PC don't look like a human face anymore but, with a little attention, is possible to see the silhouette (shape) of a dog. Even the 60-PC are better than before: the silhouette of the dog is more evident. The more are the number of PC the more easier to see becomes the silhouette of the dog. The last 6-PC is really bad, as expected, and it's not possible to understand nothing.

V. PC Visualization

I assigned different label (colours) to each type of subject: dogs are blue, guitars are green, houses are red and people are yellow. After extracting the n principal components from the matrix and fitting the model I visualized the PC.

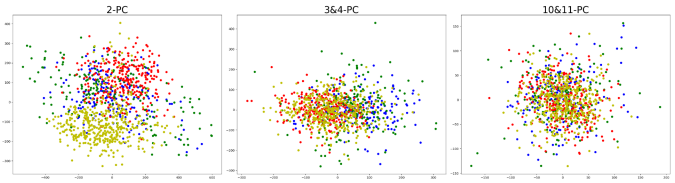

Fig. 3. *PC visualization*. Respectively from the left to the right: 2-PC, 3&4-PC and 10&11-PC

To show better what is happening I decided to do one more PC visualization increasing the number of PC.

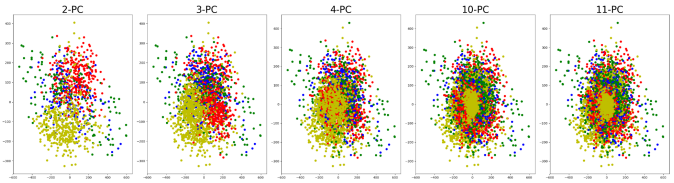

Fig. 4. *PC visualization*. Respectively from the left to the right: 2-PC, 3-PC, 6-PC, 10-PC and 11-PC.

As you can see on Fig. 3 the 2-PC and the 3-PC are quite similar and it's possible to distinguish clearly two classes: the red (houses) and the yellow (people). The dogs (blue) are not very visible and it's a little difficult to clearly separate this class from the other ones. Finally, the guitars (green) are not very visible.

In the 6-PC the two previous main classes are now overlapped. There is no more a clear difference between the two classes.

The 10-PC and the 11-PC are very very similar among them and all the classes seems like they are all overlapped. This is due to the fact that now the number of PC is higher than before and it means that more information are brought. As you can notice, the overlapping start from the first 6-PC and with the 10-PC and the 11-PC is even more visible. This is due to the fact that the first PC has the largest possible *variance*; so, the difference between the classes is more evident. Now, the PC brings with them more information and so, in addition to the PC with the largest variance, also the PC with the lowest variance: this causes the impossibility of being able to notice a clear difference between the classes.

So, *how decide the number of PC needed to preserver data without much distortion?* It depends on the task I have to perform: if I have to do image reconstruction the more are the number of PC the more 'similar' to the original image will be the reconstructed on; if i have to visualize the PC the less are the PC the more visible will be the difference between classes.

VI. Naive Bayes Classifier

Naive Bayes classifiers are a family of simple "probabilistic classifier": a classifier that is able to **predict**, given an observation of an input, a probability distribution over a set of classes, rather than only outputting the most likely class that the observation should belong to [2].

The Naive Bayes classifier is based on applying **Bayes' theorem** with strong (naive) independence assumptions between the features.

$$\frac{p(C_k|x) = p(C_k)p(x|C_k)}{p(x)}$$

After splitting the data-set into train and test set, I used the Naive Bayed Classifier in several cases and I check the respective accuracy.

TABLE I
Data classification

Case	Succes rate
Unmodified images	74.65%
2-PC	59.05%
3-PC	63.23%
4-PC	64.62%

In Table I is possible to notice that the higher are the number of PC the more is the accuracy. This is due to the application of the PCA: it allows estimating probabilities in high dimensional data but with a dramatic reduction in size of data. This reduction could be a good thing in some cases, but not in this one. Reduction means lose information and then also the classifier has less information to train; because of this, the accuracy is higher when the number of PC is high because of less information are lost. However, it can not be better than the accuracy of the original image.

VII. More

I did one more task beyond the assigned ones. I used only one photo and I considered **Lenna**. Lenna (or Lena) picture is one of the most widely used standard test images used for compression algorithms [3]. Lenna is a digitized *Playboy* centerfold, from November 1972 [4].

As anticipated I did not use an huge data-set but one single photo. After reading the image as 3-D array i reshaped it in 2-D array and then I calculated the first 50-PC. Below the 512x512 image and on the right its reconstruction using the 50-PC.

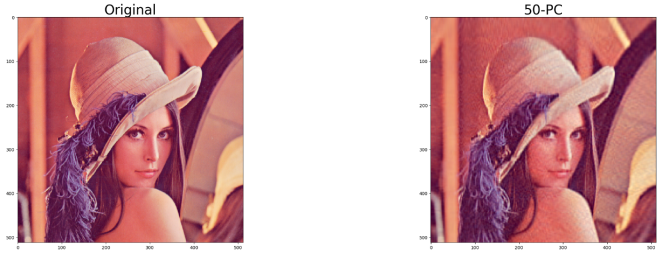

Fig. 5. 'Lenna' (or 'Lena') picture on the left. On the right its 50-PC reconstruction

In this case the reconstructed image is a blurred image of the original one.

VIII. Conclusions

A. PCA

One of the most important applications of PCA is for speeding up machine learning algorithms. It use useful to estimate probabilities in high dimensional data thanks to its dramatic reduction in size of data. On the other hand it can be too expensive for any application and can only capture linear variation.

B. Naive Bayes Classifier

It is a simple technique for constructing classifiers and it is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. It can be used for real time prediction because it's fast. A limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

References

[1] From Wikipedia, Principal component analysis. https://en.wikipedia.org/wiki/Principal_component_analysis
[2] From Wikipedia, Naive Bayes Classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[3] "Playboy centrefold photo shrunk to width of human hair". BBC News Online. 14 August 2012. Retrieved 15 August 2012. https://www.bbc.com/news/technology-19260550
[4] Lenna Story. http://www.lenna.org.