PCA CLASSIFIER REPORT

A PREPRINT

Enrico Cecchetti
Department of Computer Science
Politecnico di Torino
PoliTO, Torino

February 10, 2019

ABSTRACT

The sheer size of data in the modern age is not only a challenge for computer hardware but also a main bottleneck for the performance of many machine learning algorithms. The main goal of a PCA analysis is to identify patterns in data; PCA aims to detect the correlation between variables. If a strong correlation between variables exists, the attempt to reduce the dimensionality only makes sense. In a nutshell, this is what PCA is all about: Finding the directions of maximum variance in high-dimensional data and project it onto a smaller dimensional subspace while retaining most of the information.

1 Introduction

Idea: Given some dataPoints in a certain d-dimentional space, project them into a lower dimentional space while preserving as much information as possible.
Some examples:
Find the best planar approximation to a 3-D image.
Find the best 12-D approximation to a 10^14 data.
In particular to chose a projection that **maximize squared error** this is the factor that permits us to catch the more info possible.

2 Definition

Given a certain dataset, PCA is an algorithm that permits an orthogonal projection of the data onto a lower-dimentional linear space

- that maximize data of the projected data;
- minimize min squared distance between data-point and projection

3 Approches

There are 2 main approches for the PCA:

- Sequential (through PC vectors)
- Through Covariance Matrix
- SVD of data matrix

3.1 PCA through PC-vectors (principal components vectors)

PCA permits the transportation of data onto a lower dimension by computing the k-PC vectors of the selected dataset, and using only a set of them to reproject the image.
Properties:

- Those vectors originate from the center of mass;
- The first PC(1) point on the direction of the largest variance;
- Any other is orthogonal of the first one point on the largest variance in the residual data-space (the more we extract the less information we get);

3.2 PCA through Covariance Matrix

Given a Dataset { x1...xn }, it consist on calculate the sample Covariance Matrix by subtracting for each row xi the mean value (*standardization*) obtaining X, then the **covariance matrix** would be equal to

$$Cov = X * \underline{X} \tag{1}$$

(where $\underline{X}$ is the transposition of X).
The remaining task will be to obtain from this matrix the set of **eighenvalues** and **eighenvectors**.

$$\{\lambda i, \mu i\} \tag{2}$$

The eighenvector of the cov-Matrix are the basic component of the PCA, and the more the associated eighenvalue is larger the more important is the associated eighenvector. I mean, the more the eighenvalue, the more the eighenvector maximize the variance (so the more would be the info that we are gathering about the model).
So, after obtain them we have to sort them in Descending order and take the set base onto the number of components we want to take in consideration to reduce the data-space (*top-k eighenvectors*).

$$Ordered\{\lambda 1 > \lambda 2 > ... > \lambda n\} \tag{3}$$

$$Takethetop(first)k = 50ofthem. \tag{4}$$

This is also the technique adopted in the sketch for the calculation of the last 6 PCs.

3.2.1 PCA from Scratch

This is an extract from the code that i had post.

```
import numpy as np
def pca_from_scratch(n_components, first_or_last, std_matrix):
    cov_X = np.cov(std_matrix)
    eig_val_cov, eig_vec_cov = np.linalg.eig(cov_X)

    for ev in eig_vec_cov:
        np.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))

    # Make a list of (eigenvalue, eigenvector) tuples
    eig_pairs = [(np.abs(eig_val_cov[i]), eig_vec_cov[:, i])
                 for i in range(len(eig_val_cov))]

    # Sort the (eigenvalue, eigenvector) tuples
    if first_or_last is True:
        eig_pairs.sort(key=lambda x: x[0], reverse=True)
    else:
        eig_pairs.sort(key=lambda x: x[0], reverse=False)

    matrix_w = np.hstack([eig_pairs[i][1].reshape(1087, 1)
                 for i in range(n_components)])
    return matrix_w
```

This code is exactly the 2nd Algorithm that I explaned before, the one with the sample covariance matrix to be clear.
It could be devided in 5 steps:

- standardize matrice X
- compute covariance matrix
- extract eighenvalue and eighenvector from covariance matrix
- sort tuples (eighenvalue, eighenvector) by the eighenvalues (ascendent)
- compute matrix with first k PCs

3.2.2 Standardize matrice of samples X

We can avoid this passage because we just have as input that one. However the function to use to standardize it is:

```
# standardization of feature matrix
# makes feature_matrix with mean 0 & variance 1 -> as definition of standardization
std_X = (X - np.mean(X)) / np.std(X)
```

Just for clarification, standardization is the process of putting different variables on the same scale.
Otherwise, variables measured at different scales do not contribute equally to the analysis.
For example, in boundary detection, a variable that ranges between 0 and 100 will outweigh a variable that ranges between 0 and 1. Using these variables without standardization gives the variable with the larger range a weight of 100 in the analysis.

3.2.3 Compute covariance matrix

Covariance matrix (also known as dispersion matrix or variance–covariance matrix) is a matrix whose element in the i, j position is the covariance between the i-th and j-th elements of a random vector.
Intuitively, the covariance matrix generalizes the notion of variance to multiple dimensions.

As an example, the variation in a collection of random points in two-dimensional space cannot be characterized fully by a single number, nor would the variances in the x and y directions contain all of the necessary information; A 2x2 matrix would be necessary to fully characterize the two-dimensional variation.
for this operation I used a bases instruction of the numpy lib:

```
cov_X = np.cov(std_matrix)
```

3.2.4 Extract eighenvalue and eighenvector from covariance matrix

for extract the eighenvalue and eighenvector from the covariance matrix I used the instruction:

```
eig_val_cov, eig_vec_cov = np.linalg.eig(cov_X)
```

The eigenvectors represent the directions or components for the reduced subspace of B, whereas the eigenvalues represent the magnitudes for the directions.
The eigenvectors can be sorted by the eigenvalues in descending order to provide a ranking of the components or axes of the new subspace. From this point the next point

3.2.5 Sort tuples (eighenvalue, eighenvector) by the eighenvalues

As previously mentioned the next step would be to sort the pairs eighenvalue and eighenvector in descending orde. This will give us a rank starting from the top principal component for reprojecting the image.
About that, note that we can also choose to sort them in Ascending order as we did in this piece of code:
for this operation I used a bases instruction of the numpy lib:

```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_val_cov[i]), eig_vec_cov[:, i])
             for i in range(len(eig_val_cov))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
# reverse=False == Ascending, so from the one with highest variance
to minimum
if first_or_last is True:
    eig_pairs.sort(key=lambda x: x[0], reverse=True)
else:
    eig_pairs.sort(key=lambda x: x[0], reverse=False)
```

The fact of having a variable *first or last* permits me to sort and take the last 6 eighenvector instead of the first 6. In this extract at first we combine every eighenvector to his own eighenvalue in a list of tuple, then we sort this one based on a parameter in input.
As, it's written in the comment below we have a Descending(normal) order in case we pass False as parameter, Ascending otherwise.

3.2.6 Compute matrix with first k PCs

A total of m or less components must be selected to comprise the chosen subspace. Ideally, we would select k eigenvectors, called principal components, that have the k largest eigenvalues.

```
matrix_w = np.hstack([eig_pairs[i][1].reshape(1087, 1)
             for i in range(n_components)])
```

This istruction extracts from every touple the corresponding eighenvector, transpose it in (1087*1).
All those eighenvector are just listed as [ev[1]...ev[n]] at the end of this phase, so we have only to stack those in sequence horizontally (column wise) to end this phase.
After this phase we have the matrix_w ready to be plotted. So, we got our data in a reducted dataspace, ready to be used for our purpose.

4 Uses and problem

The algorithm that we have summarize is very usefull for a bunch of stuff, like:

- Get a compact description of the data, summarize them decreasing the dimentionality
- Ignore noise, infact taking the k-PCs decrese the noise inserted in the image
- Improve classification (Hopefully)

At contrary it can also create some kind of problem. One of them can be easily seen in the example at 1 .

This mistake refersthe case we take the whole dataset, that contains image of various class toghether, what we risk in applying the PCA is that:
If there are lot more image of one class with the respect to the other, could happen that the main k-PCs will be the one that mainly characterize that class. It means that for example, our image that is from another class will be mainly recostructed with PCs suited for other class - So the projection will be very different respect to the real one.
Other limitation of PCA are:

- Relies on linear asumptions
  PCA is focused on finding orthogonal projections of the dataset that contains the highest variance possible in order to 'find hidden LINEAR correlations' between variables of the dataset. This means that if you have some of the variables in your dataset that are linearly correlated, PCA can find directions that represents your data.
 But if the data is not linearly correlated (f.e. in spiral, where x=t*cos(t) and y =t*sin(t)), PCA is not enough
- Relies on orthogonal tranformations
  Sometimes consider that principal components are orthogonal to the others it's a restriction to find projections with the highest variance

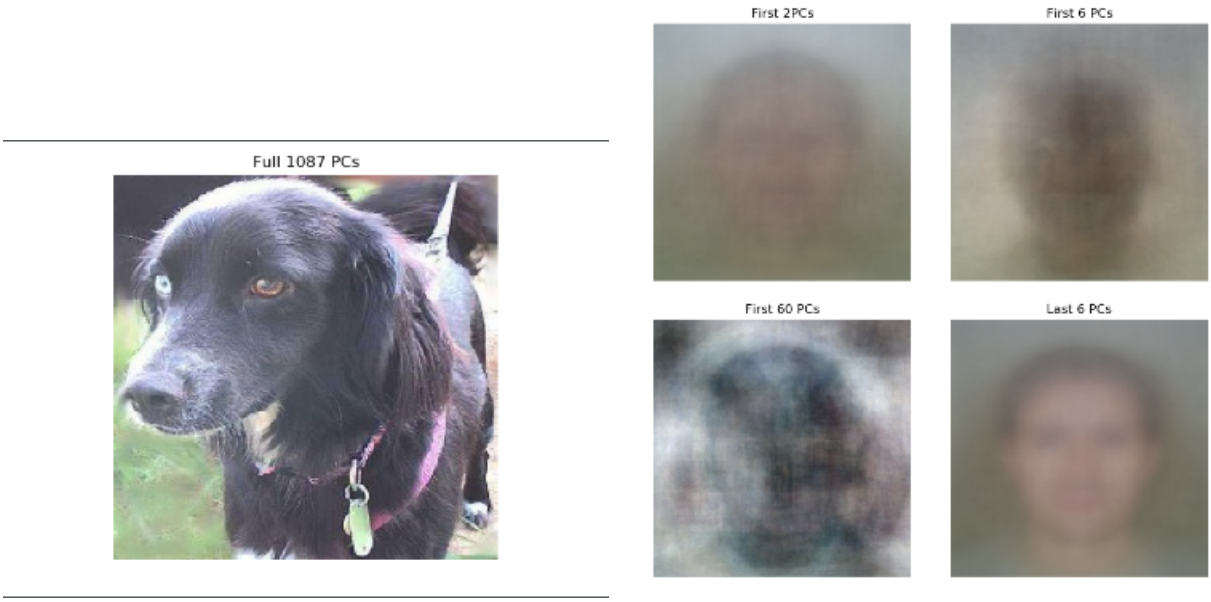

Figure 1: Dataset mistake.

- Scale variant
  PCA, as you could've seen, is a rotation trasnformation of your dataset, wich means that doens't affect the scale of your data. It's worth to said also that in PCA you dont normalize your data. That means that if you change the scale of just some of the variables in your data set, you will get different results by applying PCA.

References

[1] Scikit learn sklearn.decomposition.PCA https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[2] Ploty Principal Component Analysis in Python https://plot.ly/ipython-notebooks/principal-component-analysis/

[3] Jake VanderPlas In Depth: Principal Component Analysis In *Python Data Science Handbook*, 2018.

[4] Barbara Caputo (PCA)Principal Component Analysis In *Slides*, 2018.