Principal Component Analysis

Alessia Mondolo

October 31, 2019

Akademy.AI

Introduction to PCA

Dimensionality reduction

Goal: reducing the number of variables under consideration by obtaining a set of principal variables.

How does it work?: Transforming the data in the high-dimensional space to a space in fewer dimensions:

- PCA: not considering labels, **unsupervised**
- LDA: considering labels, **supervised**

Usage: avoiding curse of dimensionality or overfitting due to strong correlations.

In essence, it allows us to keep the "essence" of our data while greatly **reducing the complexity and dimension of the space**.

Principal Component Analysis

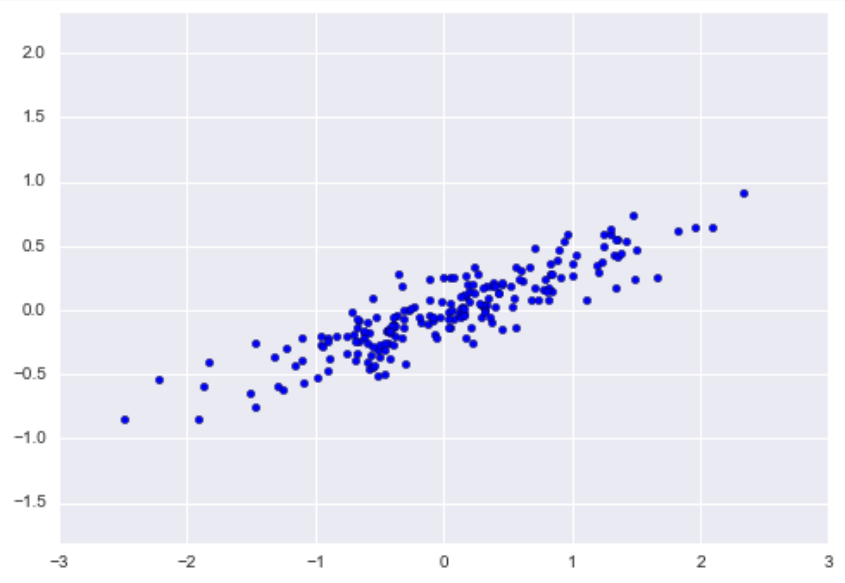
Principal Component Analysis (PCA) is perhaps one of the most broadly used of unsupervised algorithms.

PCA is fundamentally a **dimensionality reduction algorithm**, but it can also be useful as a tool for **visualization**, for **noise filtering**, for **feature extraction and engineering**, and much more.

In other words, principal component analysis is a **fast and flexible unsupervised method** for dimensionality reduction in data.

PCA: example

Consider the following 200 points:



By eye, it is clear that there is a nearly linear relationship between the `x` and `y` variables. Supervised algorithms (like linear regression), try to predict the `y` values from the `x` values, while the unsupervised learning problem attempts to learn about the **relationship** between the `x` and `y` values.

In PCA, this relationship is quantified by finding a list of the **principal axes** in the data, and using those axes to **describe** the dataset.

PCA: example

Using Scikit-Learn's `PCA` estimator, we can compute it as follows:

```
from sklearn.decomposition import PCA
pca =PCA(n_components=2)
pca.fit(X)
```

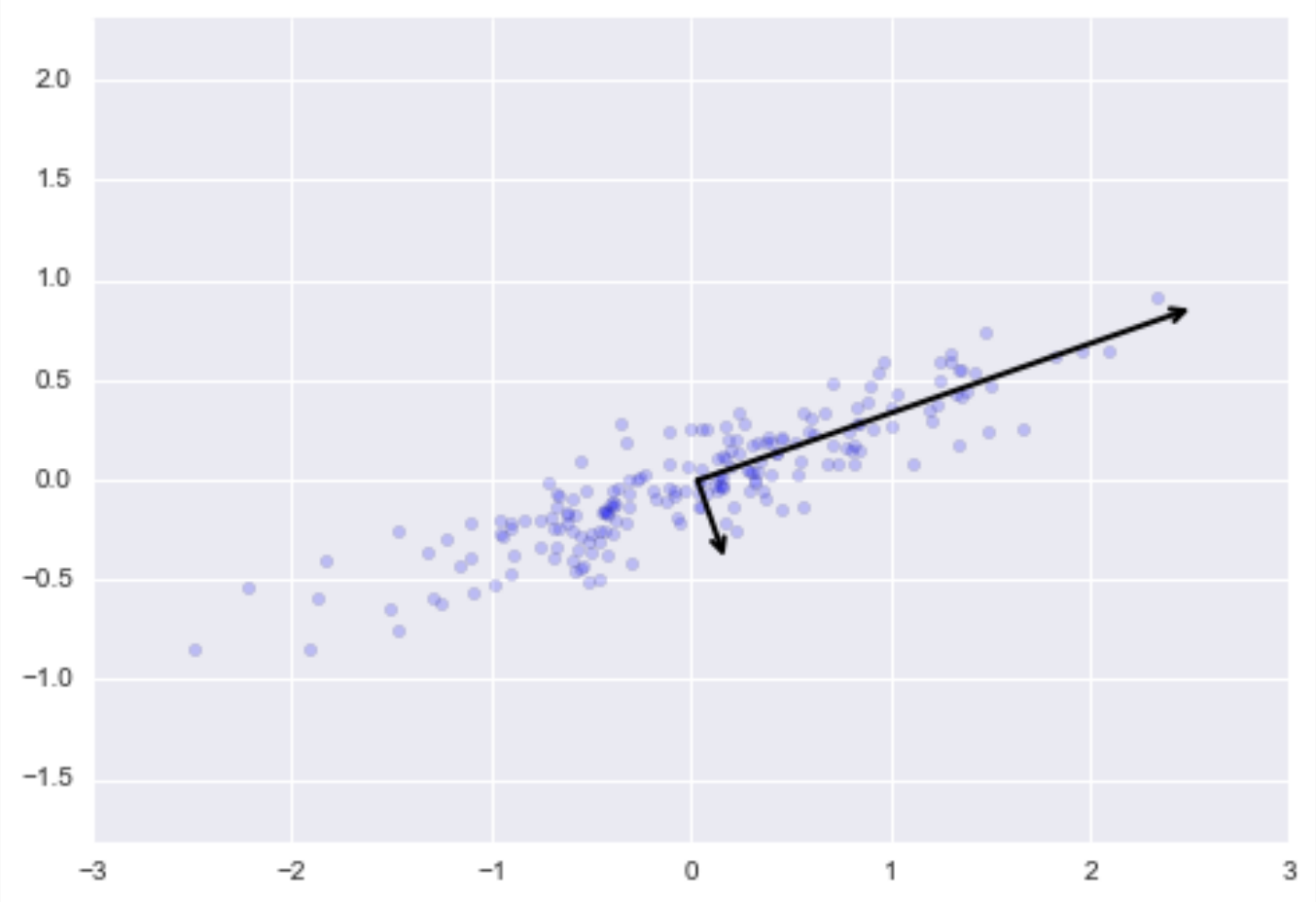
The `fit` learns some quantities from the data, most importantly the *components* and *explained variance*:

```
print(pca.components_)

[[0.94446029 0.32862557]
 [0.32862557 -0.94446029]]
```

```
print(pca.explained_variance_)

[0.75871884 0.01838551]
```

PCA: example

To see what these numbers mean, let's visualize them as **vectors** over the input data, using the *components* to define the direction of the vector, and the *explained variance* to define the squared-length of the vector:



PCA: overview

These vectors represent the **principal axes of the data**, and the length of the vector is an indication of how *important* that axis is in describing the distribution of the data — more precisely, it is a **measure of the variance of the data** when projected onto that axis. The projection of each data point onto the principal axes are the **principal components** of the data:

- The first principal component is the direction of more variability (covariance) in the data.
- The second principal component is orthogonal to the direction of the first (no correlation).

These new variables (there can be 3,4,5... principal components):

- Are **linear combinations** of the first ones;
- Have no relationship amongst them, since they are **orthogonal** in the original space;
- Capture up to 100% of the variance as the original data (usually approx. 90%).

PCA: the algorithm

PCA: the algorithm

1. Compute the mean feature vector

   $\mu = \frac{1}{p}\sum_{k=1}^{p} x_k$, where, x_k is a pattern ($k = 1$ to p), p = number of patterns, x is the feature matrix

2. Find the covariance matrix

   $C = \frac{1}{p}\sum_{k=1}^{p}\{x_k - \mu\}\{x_k - \mu\}^T$ where, T represents matrix transposition

3. Compute Eigen values λ_i and Eigen vectors v_i of covariance matrix

   $Cv_i = \lambda_i v_i$ $(i = 1, 2, 3,....q)$, q = number of features

4. Estimating high-valued Eigen vectors

   (i) Arrange all the Eigen values (λ_i) in descending order

   (ii) Choose a threshold value, θ

   (iii) Number of high-valued λ_i can be chosen so as to satisfy the relationship

   $$\left(\sum_{i=1}^{s}\lambda_i\right)\left(\sum_{i=1}^{q}\lambda_i\right)^{-1} \geq \theta$$, where, s = number of high valued λ_i chosen

   (iv) Select Eigen vectors corresponding to selected high valued λ_i

5. Extract low dimensional feature vectors (principal components) from raw feature matrix.

   $P = V^T x$, where, V is the matrix of principal components and x is the feature matrix

Algorithm translated to English

1. Subtract to each variable its mean for all the dataset
2. Calculate the covariance matrix
3. Calculate the eigenvalues and the eigenvectors of the covariance matrix
4. Pick the components (according to the criteria) and build the new matrix with which we transform the data
5. Transform the new dataset

Algorithm: step 1

Subtract to each variable its mean for all the dataset:

Data =

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Mean: 1.81 1.91

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Mean: 0 0

Algorithm: step 2 and 3

Calculate the covariance matrix (`np.cov(x)`):

$$cov = \begin{matrix} \\ x \\ y \end{matrix} \begin{matrix} \begin{matrix} x & y \end{matrix} \\ \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix} \end{matrix}$$

Calculate the eigenvalues and the eigenvectors (`np.linalg.eig(a)`):

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$

Algorithm: step 4 and 5

Multiply all the original dataset by the new matrix:

$$TransformedData = RowFeatureVector \times RowDataAdjust$$

$$RowFeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

$$RowFeatureVector2 = \begin{pmatrix} -.677873399 & -.735178956 \end{pmatrix}$$

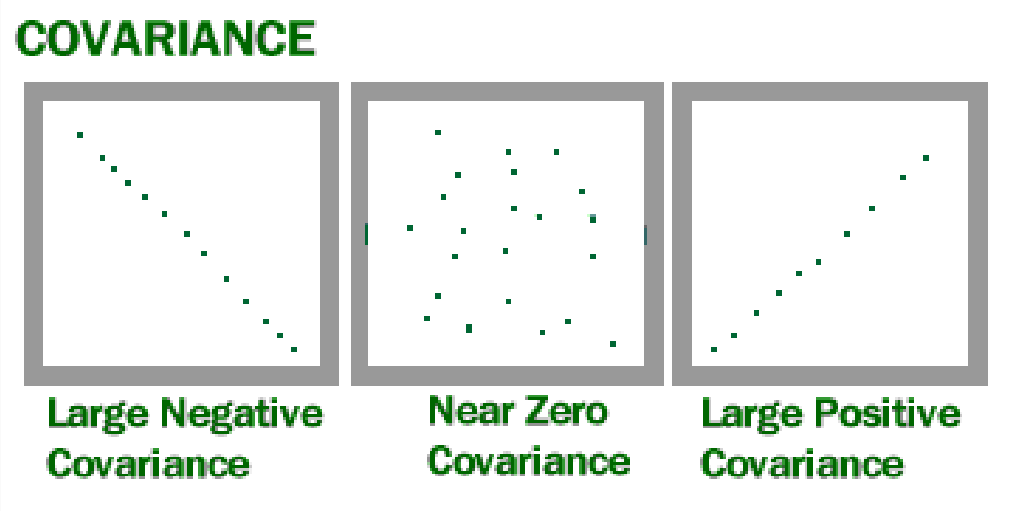
$$RowDataAdjust = \begin{pmatrix} .69 & -1.31 & .39 & .09 & 1.29 & .49 & .19 & -.81 & -.31 & -.71 \\ .49 & -1.21 & .99 & .29 & 1.09 & .79 & -.31 & -.81 & -.31 & -1.01 \end{pmatrix}$$

Covariance

The Covariance of two attributes is defined as:

$$\mathrm{cov}(A_1, A_2) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

- **Positive:** Both variables grow
- **Negative:** One grows, the other decreases
- **Zero:** Variables are independent



Covariance matrix

It is the covariance of each of the variables of a dataset against all the others.

Example of a dataset with variables H & M:

$$C^{n\times n} = (c_{i,j}), \text{where } c_{i,j} = \text{cov}(A_i, A_j)$$

$$\begin{pmatrix} \text{cov}(H,H) & \text{cov}(H,M) \\ \text{cov}(M,H) & \text{cov}(M,M) \end{pmatrix}$$

Covariance != Correlation

- **Correlation:** a measure used to represent how strongly two random variables are related
- **Covariance:** a measure used to indicate the extent to which two random variables change in tandem.
- **Covariance** is in a way, a measure of correlation
- **Correlation** goes **from -1 to 1**
- **Covariance** goes **from** $-\infty$ **to** $+\infty$.
- **Covariance** is affected by a change in scale. Correlation is not.

PCA vs. LDA

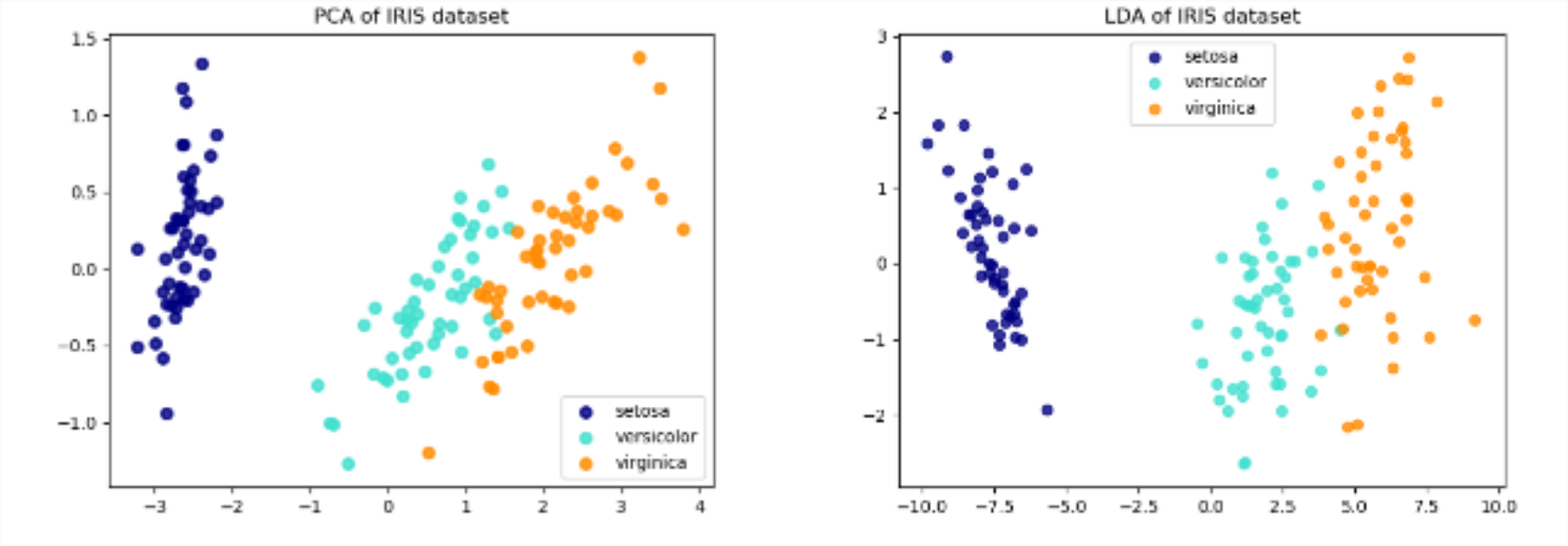
Principal Component Analysis (PCA) identifies the combination of attributes (principal components, or directions in the feature space) that account for the most variance in the data.

Linear Discriminant Analysis (LDA) tries to identify attributes that account for the most variance between classes. In particular, LDA, in contrast to PCA, is a supervised method, using known class labels.

PCA seeks to find the direction that maximizes **intra-cluster** variance. The idea is to project the cluster along a dimension such that **all the data points are very well separated**.

LDA seeks to find a direction that maximizes **inter-cluster** variance. The idea is to **make different sets of data as distinguishable as possible**.

PCA vs. LDA: iris dataset



PCA as dimensionality reduction

Dimensionality reduction

Goal: reducing the number of variables (dimensionality) by maximizing the explained variance.

We'd like to pick a number of components that can explain up to 90% of the variance in the data.

Advantages:

- Reducing the number of variables, more simple representation
- Easier regression analysis

Disadvantages:

- More difficult to explain the meaning
- We fundamentally "miss" some data

How to reduce dimensions

Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

Here is an example of using PCA as a dimensionality reduction transform:

```
pca =PCA(n_components=1)
pca.fit(X)
X_pca =pca.transform(X)
print("original shape: ", X.shape)
print("transformed shape:", X_pca.shape)
```

Output:

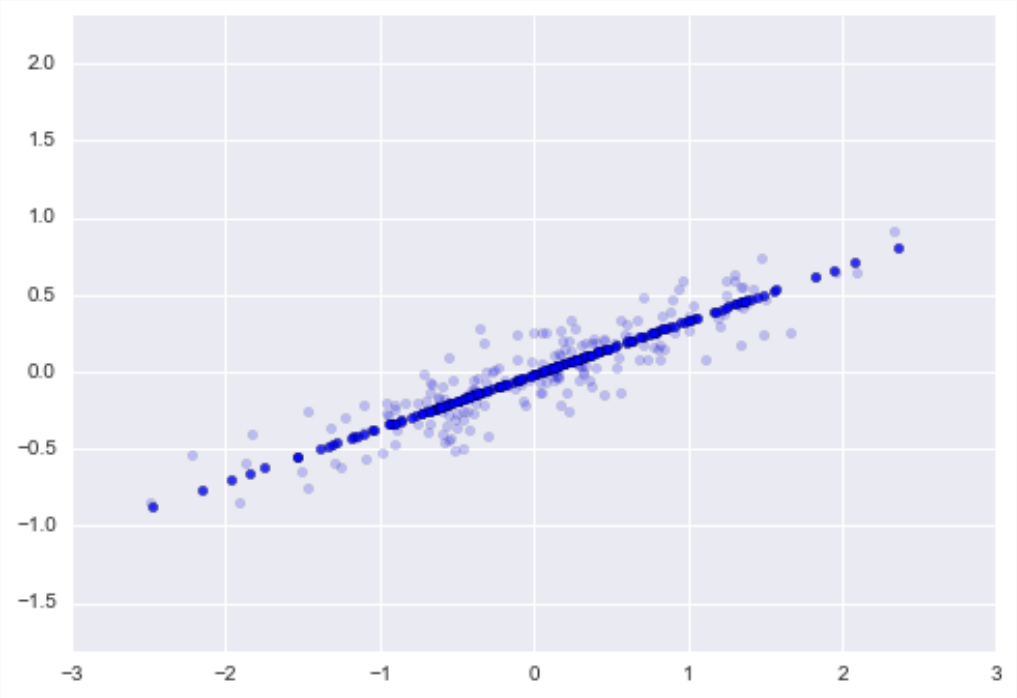
```
original shape:  (200, 2)
transformed shape:  (200, 1)
```

The transformed data has been reduced to a single dimension.

How to reduce dimensions

To understand the effect of this dimensionality reduction, we can perform the inverse transform of this reduced data and plot it along with the original data:

```
X_new =pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```



PCA: other applications:

- PCA for Visualization
- PCA as Noise Filtering

You will find some examples of these applications in the first notebook for today.