Clustering with Linear Discriminant Analysis (LDA)

Posted Sep 26, 2022

By Davide Evangelista

8 min read

Why PCA is not sufficient?

Since PCA is an unsupervised learning technique, the lack of informations due to the fact that it is not using the associated label reduce its ability on observing clusters. Consequently it can be defined Linear Discriminant Analysis (LDA), a supervised version of PCA, whose ability on generating clusters is way higher.

Setup for LDA implementation

The main difference between PCA and LDA is that in the latter we center the dataset by shifting each column by the centroid of each class, instead of using a global centroid for the data. Unfortunately, this slight modification will make the algorithm way more technical than before. For simplicity, assume we have a dataset

$$X = [x^1 x^2 \dots x^N] \in \mathbb{R}^{d \times N}$$

and an associated label vector

$$Y = [y^1 y^2 \dots y^N]$$

such that y^i represents the correct class of x^i. Assume that in our problem there are K classes $C_1, \dots, C_K$ and define $I_k = [i_1, \dots, i_{N_k}]$ as the vector of the indices corresponding to the columns of X associated with the class k (N_k is the number of elements lying in the class k). Clearly, it holds that $N_1 + N_2 + \dots + N_K = N$. To simplify the discussion, assume that our dataset is ordered as

$$X = [X_1 X_2 \dots X_K]$$

where X_k contains each datapoint such that the corresponding class is k. Now, for each class $k = 1, \dots, K$ define the class centroid

$$c_k(X) = \frac{1}{N_k} \sum_{i \in I_k} x^i$$

and the global centroid

$$c(X) = \frac{1}{N} \sum_{i=1}^{N} x^i$$

We can now define the main tools for our algorithm: the **within-clusters** and **between-clusters** scatter matrices.

Within-clusters Scatter Matrix

For any $k = 1, \dots, K$, define the centered matrix for each class

$$X_{k,c} = X_k - c_k(X_k)$$

and, by concatenating $X_{k,c}$ for any k, we get

$$X_w = [X_{1,c} X_{2,c} \dots X_{K,c}]$$

From X_w, we can define the **within-cluster scatter matrix**

representing the *correlation matrix* for points inside of each cluster.

Between-clusters Scatter Matrix

Not, define

$$\bar{X}_k = [c_k c_k c_k \dots c_k] \in \mathbb{R}^{d \times N_k}$$

$$\bar{X} = [\bar{X}_1 \bar{X}_2 \dots \bar{X}_K]$$

and its centered version

$$\bar{X}_c = \bar{X} - c(X)$$

Finally, the **between clusters scattering matrix** is

$$S_b = \bar{X}_c \bar{X}_c^T$$

This concludes the setup part.

Idea

Given the quantities defined above, the idea of LDA is to build a projector $Q \in \mathbb{R}^{d \times k}$ such that the within-clusters distance of the projected data is as small as possible, while the between-clusters distance of the projected data is as big as possible. If $k \in \mathbb{N}$, $k < d$, is the dimension of the projected space and $Q = [q_1 q_2 \dots q_k]$ is the projector, we want q_js to be orthogonormal vectors that maximize the function

$$H(q) = \frac{q^T S_w q}{q^T S_b q}$$

Since for any $\alpha > 0$, $H(\alpha q) = H(q)$, it can be proved that maximizing $H(q)$ is equivalent to find the solution of the constraint minimization problem

$$\max_q q^T S_w q \qquad \text{s.t. } q^T S_b q = 1$$

Which can be easily solved by the following procedure. First of all, observe that S_w is symmetrical, since $S_w^T = (X_w X_w^T)^T = X_w X_w^T = S_w$. If S_w is also positive definite, it is known that it is possible to compute its Cholesky decomposition as

$$S_w = L L^T$$

where $L \in \mathbb{R}^{d \times d}$ is a lower-triangular matrix.

If S_w is not positive definite, we can make it positive definite by doing

$$S_w = S_w + \epsilon I$$

where $\epsilon \approx 10^{-6}$ is a small value. A matrix like that will always be positive definite.

Now, given a number k (representing the dimension on which we want to project), compute the matrix $W \in \mathbb{R}^{d \times k}$, whose column are the first k eigenvectors of $L^{-1} S_b L$.

Finally, if

$$Q = L^{-T} W$$

then $Q^T \in \mathbb{R}^{k \times d}$ will be the projection matrix of LDA.

The implementation simply follows what we just described.

- Given the input data X and the labels Y;
- Compute S_w and S_b as described above;
- If possible, compute the Cholesky decomposition of $S_w = LL^T$, if not, compute the Cholesky decomposition of $S_w + \epsilon I = LL^T$;
- Find the first k eigenvectors of $L^{-1}S_bL$ and collect them in a matrix W;
- Compute $Q = L^{-T}W$ and Q^T.
- Since Q^T is the projection matrix, we can project X into Z as $Z = Q^TX$.

Python code (saw in class)

First of all, we import the MNIST dataset. As always, in the example we are going to use the `data.csv` file we used in the past labs. It can be found on Virtuale or downloaded from Kaggle.

Plaintext

```
# Import the data (MNIST) as always
import numpy as np
import pandas as pd

# Load data into memory
data = pd.read_csv('data.csv')

print(data.shape)
```

After that, the imported data has to be splitted into its input and output components, X and Y, after we converted it into a numpy array. Moreover, to simplify the computation, we will split both X and Y into three components, containing all the datapoints from the classes of digits 0, 6 and 9.

Plaintext

```
Split the data into:
# - X, Y
# - Isolate X1, X2, X3 from X, where:
#       - X1 is the sub-data of X containing 0
#       - X2 is the sub-data of X containing 6
#       - X3 is the sub-data of X containing 9
# - Split Y into Y1, Y2, Y3 accordingly

# Convert data into a matrix
data = np.array(data)

X = data[:, 1:]
X = X.T
Y = data[:, 0]

d, N = X.shape

# Find the corresponding indeces
I1 = (Y==0)
I2 = (Y==6)
I3 = (Y==9)

# Split X and Y into X1, X2, X3 and Y1, Y2, Y3
X1 = X[:, I1]
X2 = X[:, I2]
X3 = X[:, I3]

Y1 = Y[I1]
Y2 = Y[I2]
```

And we concatenate the components to sort the dataset by classes (following the first assumption in the theoretical part).

Plaintext

```
# Concatenate the data
X = np.concatenate((X1, X2, X3), axis=1)
Y = np.concatenate((Y1, Y2, Y3))
```

Now, we compute the within-cluster centroid $c_k(X)$ for any k and the global centroid $c(X)$.

Plaintext

```
# Within-clusters centroid
C1 = np.mean(X1, axis=1)
C2 = np.mean(X2, axis=1)
C3 = np.mean(X3, axis=1)

# Global centroid
C = np.mean(X, axis=1)
```

After that, we compute the within-cluster scatter matrix S_w and the between-cluster scatter matrix S_b, following the theoretical part.

Plaintext

```
# Center each cluster dataset
X1c = X1 - C1.reshape((d, 1))
X2c = X2 - C2.reshape((d, 1))
X3c = X3 - C3.reshape((d, 1))

# Compute the within-cluster matrix by concatenation
Xw = np.concatenate((X1c, X2c, X3c), axis=1)

# Compute the within-cluster scatter matrix
Sw = Xw @ Xw.T

# Compute the Xbars
Xbar1 = np.repeat(C1.reshape(d, 1), X1.shape[1], axis=1)
Xbar2 = np.repeat(C2.reshape(d, 1), X2.shape[1], axis=1)
Xbar3 = np.repeat(C3.reshape(d, 1), X3.shape[1], axis=1)

# Compute the between-cluster dataset
Xbar = np.concatenate((Xbar1, Xbar2, Xbar3), axis=1)

# Compute the between-cluster centered dataset
Xbarc = Xbar - C.reshape((d, 1))

# Compute the between-cluster scatter matrix
Sb = Xbarc @ Xbarc.T
```

To go on, if S_w is symmetric positive definite, compute its Cholesky decomposition $S_w = LL^T$. If it is not positive definite, then we compute the Cholesky decomposition of $S_w + \epsilon I$, for $\epsilon = 10^{-6}$.

Plaintext

```
# We want to compute the Cholesky decomposition of Sw
try:
    L = np.linalg.cholesky(Sw)
except:
    epsilon = 1e-6
    Sw = Sw + epsilon * np.eye(Sw.shape[0])

    L = np.linalg.cholesky(Sw)
```

To conclude, we can import `scipy.sparse.linalg` and use the function `scipy.sparse.linalg.eig()` to compute the matrix W and consequently the projection matrix Q.

Plaintext

```
import scipy
import scipy.sparse
import scipy.sparse.linalg

# Compute the first k eigenvector decomposition of L^-1 Sb L
_, W = scipy.sparse.linalg.eigs(np.linalg.inv(L) @ Sb @ L, k=2)
W = np.real(W)

# Compute Q
Q = np.linalg.inv(L).T @ W
```

Note that, even if $L^{-1}SbL$ is symmetric and thus, as a consequence of the Spectral Theorem, its eigenvectors should be real, because of machine approximations we get complex values in W. Consequently, we need to cast it to the real numbers by the function `np.real()`, which eliminates the immaginary part of each number.

To conclude, we can use Q to compute the projection of X and plot the clusters.

Plaintext

```
# Compute the projection
Z = Q.T @ X
```

The following is the expected result.

Lab2, teaching

LDA clustering dimensionality reduction SVD unsupervised learning

Share:

Further Reading

Sep 27, 2022

Classification with clustering algorithms

What is classification? Consider, as an example, the MNIST dataset we used to te…

Sep 25, 2022

Dimensionality Reduction with PCA

Dimensionality Reduction While working with data, it is common to have access to…

Sep 24, 2022

A (very short) introduction to Machine Learning

Definition: Machine Learning (ML) is the set of all the techniques and algorithms able t…

OLDER
Dimensionality Reduction with PCA

NEWER
Classification with clustering algorithms

0 Comments - *powered by utteranc.es*

Write Preview

Sign in to comment

Styling with Markdown is supported

Sign in with GitHub