0.1 LDA

0.1.1 Abstract

In this chapter, we will learn another algorithm of the binary-classification-hard-output: LDA (linear discriminant analysis). Actually, it's also used to reduce the dimension.

We'll select a direction and project the high-dimensional samples to this direction to divide them into two classes.

0.1.2 Idea

The core idea of LDA is to make the projected data satisfy two conditions:

- the distance between samples within the same class is close
- the distance between different classes is large.

0.1.3 Algorithm

Firstly, to reduce the dimension, we have to find out how to calculate the projection length.
we assume a sample x and project it to the direction w.
As we know: $w \cdot x = ||w||\ ||x||\ \cos\theta$

Here we assume $||w|| = 1$ to detemine the unique w to prevent countless solutions caused by scaling.
so $wx = ||x|| \cos\theta$
And $||x|| \cos\theta$ is exactly the definition of projection.
Therefore, the projection length of the sample on the vector w is wx.

Thus the projection is $z = w^T x$ We assume the number of samples belonging to the two classes is $N1$,$N2$.

Below, as to the first condition: the distance of sample within the same class is close, we use the variance matrix to represent the overall distribution of each class.
Here we use the definitioin of covariance matrix and the covariance matrix of origin data x is denoted as S.

$$\begin{aligned}
C_1 : Var_z[C_1] &= \frac{1}{N_1}\sum_{i=1}^{N_1}(z_i - \bar{z_{c1}})(z_i - \bar{z_{c1}})^T \\
&= \frac{1}{N_1}\sum_{i=1}^{N_1}(w^T x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} w^T x_j)(w^T x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} w^T x_j)^T \\
&= w^T \frac{1}{N_1}\sum_{i=1}^{N_1}(x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} x_j)(x_i - \frac{1}{N_1}\sum_{j=1}^{N_1} x_j)^T w \\
&= w^T \frac{1}{N_1}\sum_{i=1}^{N_1}\left(x_i - \bar{x_{c1}}\right)\left(x_i - \bar{x_{c1}}\right)^T w \\
&= w^T S_1 w \\
C_2 : Var_z[C_2] &= \frac{1}{N_2}\sum_{i=1}^{N_2}(z_i - \bar{z_{c2}})(z_i - \bar{z_{c2}})^T \\
&= w^T S_2 w
\end{aligned} \tag{1}$$

Therefore the distance between classes can be denoted by:

$$Var_z[C_1] + Var_z[C_2] = w^T(S_1 + S_2)w$$

As to the second condition: the distance between different classes is large
The distance between classes can be denoted by the difference between the mean projection length of two classes.

$$\begin{aligned}
(z_{c1} - z_{c2})^2 &= (\frac{1}{N_1}\sum_{i=1}^{N_1} w^T x_i - \frac{1}{N_2}\sum_{i=1}^{N_2} w^T x_i)^2 \\
&= (w^T(\frac{1}{N_1}\sum_{i=1}^{N_1} x_i - \frac{1}{N_2}\sum_{i=1}^{N_2} x_i))^2 \\
&= (w^T(\bar{x_{c1}} - \bar{x_{c2}}))^2 \\
&= w^T(\bar{x_{c1}} - \bar{x_{c2}})(\bar{x_{c1}} - \bar{x_{c2}})^T w
\end{aligned} \tag{2}$$

Well, let's look back on our two conditions:

- the distance of samples within the same class is close
- the distance between different classes is large

So it's easy to obtain a intuitive loss function:

$$L(w) = \frac{Var_z[C_1] + Var_z[C_2]}{(z_{c1} - z_{c2})^2} \tag{3}$$

Via minimizing the loss function, we can obtain the target w:

$$\begin{aligned}\hat{w} = argmin(L(w)) &= argmin(\frac{Var_z[C_1] + Var_z[C_2]}{(z_{c1} - z_{c2})^2}) \\ &= argmin(\frac{w^T(S_1 + S_2)w}{w^T(\bar{x_{c1}} - \bar{x_{c2}})(\bar{x_{c1}} - \bar{x_{c2}})^T w}) \\ &= argmin(\frac{w^T S_w w}{w^T S_b w})\end{aligned} \tag{4}$$

In the formula:

$$\begin{aligned}&S_w : with - class : variance\ within\ the\ class \\ &S_b : between - class : variance\ between\ classes\end{aligned} \tag{5}$$

The following is the partial derivative of the above formula:

$$\begin{aligned}\frac{\partial L(w)}{\partial w} &= \frac{\partial}{\partial w}(w^T S_w w)(w^T S_b w)^{-1} \\ &= 2S_b w \left(w^T S_w w\right)^{-1} - 2w^T S_b w \left(w^T S_w w\right)^{-2} S_w w = 0\end{aligned} \tag{6}$$

try to transform the equation:

$$\begin{aligned}\left(w^T S_b w\right) S_w w &= S_b w \left(w^T S_w w\right) \\ \left(w^T S_b w\right) w &= S_w^{-1} S_b w \left(w^T S_w w\right)\end{aligned}$$

Notes: the shape of $w^T S_b w$ and $w^T S_w w$ is : $(1, p)(p, p)(p, 1) = (1, 1)$

Since the two terms are scalars, they only scale the module of a vector and can't change its direction, so the above formula is updated to:

$$w \propto S_w^{-1} S_b w = S_w^{-1} \left(\bar{x_{c1}} - \bar{x_{c2}}\right) \left(\bar{x_{c1}} - \bar{x_{c2}}\right)^T w$$

And because $\left(\bar{x_{c1}} - \bar{x_{c2}}\right)^T w$ is also a scalae, we obtain the final formula:

$$\hat{w} \propto S_w^{-1} \left(\bar{x_{c1}} - \bar{x_{c2}}\right)$$

So $S_w^{-1} \left(\bar{x_{c1}} - \bar{x_{c2}}\right)$ is the direction we have been seeking, finally we can get the standard w via scaling.

0.1.4 Implement

```
import numpy as np
import os
os.chdir("../")
from models.linear_models import LDA

x = np.linspace(0, 100, num=100)
w1, b1 = 0.1, 10
w2, b2 = 0.3, 30
epsilon = 2
k = 0.2
b = 20
w = np.asarray([-k, 1])
v1 = x * w1 + b1 + np.random.normal(scale=epsilon, size=x.shape)
v2 = x * w2 + b2 + np.random.normal(scale=epsilon, size=x.shape)
x1 = np.c_[x, v1]
x2 = np.c_[x, v2]
l1 = np.ones(x1.shape[0])
l2 = np.zeros(x2.shape[0])
data = np.r_[x1, x2]
label = np.r_[l1, l2]

model = LDA()
model.fit(x1, x2)
model.draw(data, label)
```