Abstract

As we know, linear classificaton is divided into two types:

- hard output (directly output the class of the samples):
  - perceptron
  - LDA(linear discriminant analysis)
- soft output(output the probability that samples belong to some class):
  - gaussian discriminant analysis
  - logistic regression

In this issue, we will introduce a simple linear binary classification model: $Perceptron$. Its requirements are relatively loose, as long as we can find a hyperplane to separate positive and negative samples.

Idea

Error driven

Literally, we can see that the idea of the perceptron model is to randomly initialize the parameters of the model, and then update its own parameters via errors according to whether the current parameters can correctly separate the positive and negative samples.

Algorithm

Firstly, given the object function of the model:

$$f(x) = sign(w^T x) \tag{6}$$

In this formula, $sign$ is a symbolic function.

$$sign(a) = \begin{cases} +1, & a > 0 \\ -1, & a \leq 0 \end{cases} \tag{7}$$

then, according to the idea of perceptron mentioned above: error driven

It's easy to obtain the loss function of the model:

$$L(w) = \sum_{i=1}^{n} I\{w^T x_i * y_i < 0\} \tag{8}$$

In this formula, I is a indicator function, indicating which elements belong to the collection.

And it's easy to understand the judgment conditions. We note that:

- when $y_i > 0$, it indicates y_i is a positive sample:
  - then, if $w^T x_i < 0$, it indicates the sample is classified incorrectly ($w^T x_i * y_i < 0$)
    - if $w^T x_i > 0$, it indicates the sample is classified correctly ($w^T x_i * y_i > 0$)
- when $y_i < 0$, it indicates y_i is a negative sample:
  - then, if $w^T x_i < 0$, it indicates the sample is classified correctly ($w^T x_i * y_i > 0$)

- if $w^T x_i > 0$, it indicates the sample is classified incorrectly ($w^T x_i * y_i < 0$)

We finally find out when $w^T x_i * y_i < 0$, it indicates the sample is classified incorrectly by model.

Well, let's look back on the loss function. We are suprised to find that it's non differentiable, thus we can't use gradient descent. Um, how can we solve the issue?

We could loose the conditions, and the loss function is updated to:

$$L(w) = \sum_{(x_i, y_i) \in M} -w^T x_i * y_i \tag{9}$$

M represents the collection classified incorrectly.

then adding a minus sign in front of $w^T x_i * y_i$, we will obtain a positive loss value. And then we can use graident descent to update the parameters w.

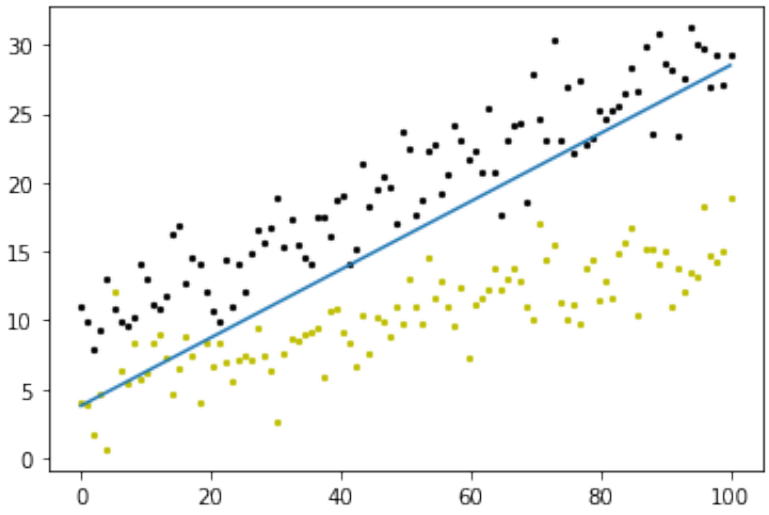
As for the derivative of the loss function, it's also easy to solve:

$$\frac{dL(w)}{dw} = \sum_{(x_i, y_i) \in M} -x_i * y_i \tag{10}$$

Implement

```
%matplotlib inline
import os
os.chdir("../")
import numpy as np
from models.linear_models import Perceptron

model = Perceptron(10000, lr=1e-2)
x = np.linspace(0, 100, num=100)
w1, b1 = 0.1, 5
w2, b2 = 0.2, 10
epsilon = 2
k = 0.15
b = 8
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]
x = np.r_[x1, x2]
y = np.sign(x.dot(w) - b)
model.fit(x, y)
model.draw(x)
```


30
25
20
15
10
5
0
0
20
40
60
80
100