Machine Learning

HW2

-Logistic Regression-

Subject : 기계학습의 기초및응용
Professor : Kye-haeng Lee
Name : Dongho Bang
Student Number : 32181995
Data : 2022.10.09

Contents

1. HW Introduction

Problem Description: Logistic Regression

The main purpose of this assignment is to code a binary classifier using logistic regression. You are given two files (train.csv and test.csv) for a dataset which is characterized by two features, denoted by x1 and x2, and each data point belongs to class 0 or 1 denoted by y. Your task is 1) to train a classifier using train.csv, and 2) to classify the data in test.csv using the trained model:

Task 1. Train a classifier with train.csv.

The file train.csv has three columns, x1, x2 and y. The first two columns indicate the two features of the data, and the last column y means the label. There are no restrictions on the decision boundary model; you can use any, but do not forget to wrap it in the sigmoid. To train your classifier, use a gradient descent algorithm as you did in the last assignment.

As a result of this task, you must give 1) the parameters of your model, 2) the cost.

Task 2. Classify the data in test.csv using the trained model.

The file test.csv contains a set of data in the same format as train.csv. Use the trained classifier in Task 1 to classify the given dataset in test.csv.

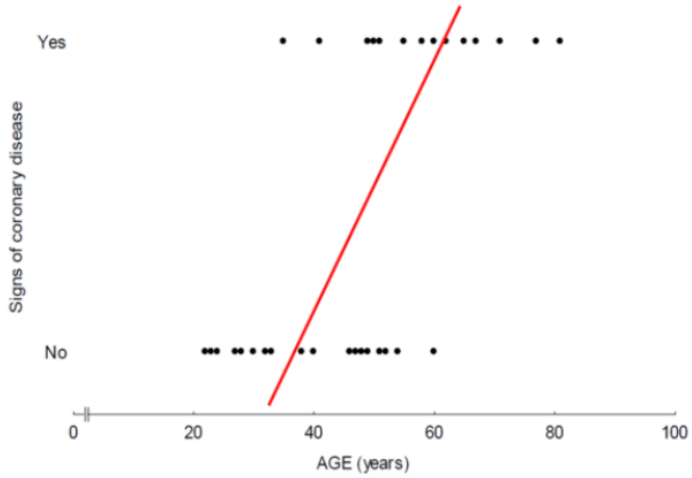
This task requires measuring two metrics: 1) the cost and 2) the accuracy which is defined as the number of correct classifications for the total number of data points.

2. Concept

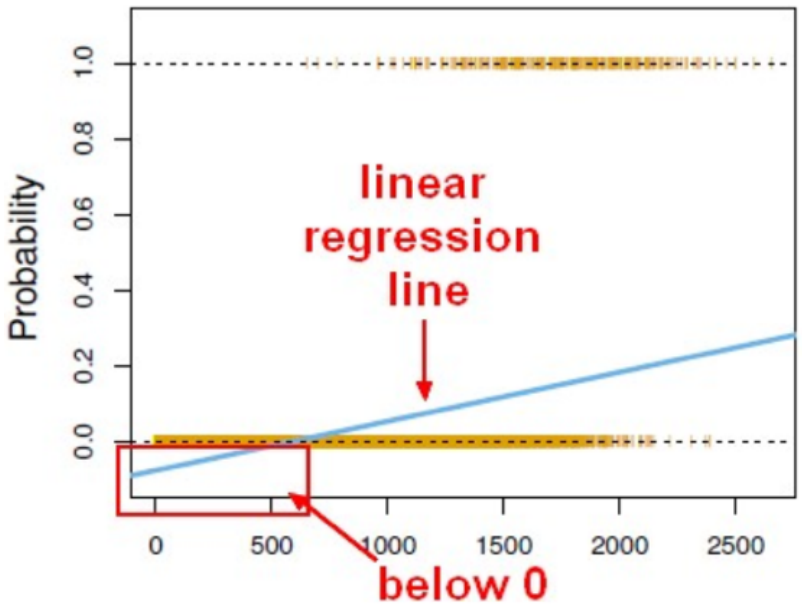
(1) 로지스틱 회귀의 이해와 Hypothesis

로지스틱 회귀 모델은 y값이 범주형 변수인 경우(binary)에도 다중선형회귀모델을 적용시키기 위해 고안된 것이다.

만약 회귀식의 결과값이 0, 1로만 도출된다면, 회귀식의 경우 이에 근사하는 직선 혹은 모델을 만들기가 어려워진다.



그래서 생각해낸 것은 타겟변수를 확률로 만들자는 것이다. 이 아이디어를 통해 (0, 1)을 0 ≤ Pr(Y=1 | X) ≤ 1로 나타내게 된다. 하지만 이 역시 문제가 발생한다. Model : P(X)= β_0+βX의 모델은 회귀선에 의해 음수값을 추정하거나, 경우에 따라 1을 초과하는 추정값이 발생할 수도 있다.



이러한 문제점 때문에, 이항분류문제를 회귀모델을 이용하여 해결하기 위해서는 확률값 p를 -∞부터 ∞까지 확장할 필요성이 있다. 그래서 Probability 대신, 승산(Odds)이라는 개념을 도입한다. 승산이란 임의의 사건 Math input error가 발생하지 않을 확률 대비 일어날 확률의 비율을 뜻한다.

Odds= (p(X))/(1-p(X))

이를 통해 계산되는 Y의 결과값은 [0, ∞]이다. 즉, 하나의 사건이 일어날 승산이 0부터

무한대까지 표현된다는 것이다. 그러면 양수는 됐고, 이제 여기에 음수의 무한대까지 표현할 방법이 필요하다. 0부터 무한대까지 나타나는 위의 결과값 odds에 로그를 씌우면, 우리가 원하던 -∞부터 ∞까지의 결과값이 표현된다. 이렇게 오즈에 로그를 취하는 것을 로짓 변환이라고 한다.

$$\text{logit}(p) = \log \frac{p}{1-p}$$

이제, 로짓 변환을 이용하여 위의 식 Model을 다시 표현하면 ln((p(X))/(1-p(X))) = β_0+βX 가 되고, 이를 다시 P(X)에 대해 정리(아래의 그림과 수식에서는 f(X)로 표현)하면 다음과 같다.

f(X) : Logistic Function

$$f(x) = \frac{1}{1+e^{-x}}$$

이를 통해 얻어낸 Hypothesis는 결과적으로, 범위가 제한되지 않은 X 입력값들을 시그모이드(로지스틱)함수를 이용하여 Y값을 [0, 1]로 압축한 것이다. 즉 Y값은 사건이 발생할 확률이 된 것이고, 여기에 0 ~ 1 사이에 값을 가지는 임계값을 설정하여 사건이 발생할 확률이 어느 이상이여야 결과를 1로 할 지 결정한 뒤, 이를 최적화하는 학습계수들을 학습하면 된다.

(2) Decision Boundary

이제 Hypothesis의 결과로 나오는 Y값이 어떤 임계값을 가져야 사건 1로 분류가 되는지를 결정해야 한다. 이는 필연적으로 X 값들이 어떤 경계를 넘어감에 따라 Y의 결과값들이 바뀌는 것에 영향을 받는다. 위의 식에서 결국 로지스틱 회귀분석은 W, b의 값들을 찾는 문제이고, 따라서 임계값을 결정하는 하이퍼플레인 구분자, 디시전 바운더리 함수는 아래의 수식이 된다.

z = β_0+βX

==========여기부터==========

(3) 비용 함수 정의

비용함수는 W, b의 값을 찾기 위한 함수이다. 비용함수는 일반적으로 경사 하강법(Gradient Descent)를 사용하기에 적합한 함수 모양으로 만들어야 한다. 일반적인 Regression의 비용함수는 비용함수를 (예측값-실제값의 제곱의 합)과 같이 2차 함수의 형태로 만들어 GD를 적용한다. 하지만 Logistic Regression의 비용함수는 다르게 도출해낸다. 우선 자연상수 e 때문에, 예측값과 실제값의 차이의 제곱의

합과 같은 수식은 매끈한 2차함수의 모양을 띠지 않는다. 이를 상쇄하기 위해 log를 씌워주는 과정이 필요하다.

즉, 원래의 Regression의 cost function이 [Cost = Sum((예측된 y값 - 실제 y값) ^ 2) / n] 이었다면, 매끈한 비용함수를 위하여 이 식을 보정해 주어야 한다. 이 때, 단순히 log를 씌워 보정해주는 것이 아닌, Y의 결과값 1과 0을 구분하여 수식을 만든다. 먼저, Y가 1인 경우에는 아래를 따른다.

cost = (1/n) * sum (-hypothesis) = (1/n) * sum (-log(sigmoid(Wx+b)))

이를 해석해보면, Hypothesis의 예측값이 0에 가까워질수록 cost의 값이 커지는 것을 알 수 있다. 이는 cost의 최저값을 찾는 모델 학습에 부합하는 개념이다. 다음으로 Y가 0인 경우에는 cost = (1/n) * sum(-log(1-sigmoid(Wx+b)))의 수식을 따른다. 이 수식 역시 예측값이 부정확할수록 cost가 커지는 것을 알 수 있다. 수학적으로 이 두가지를 하나의 수식으로 합치면(이 영역은 수학자의 영역이니 이해하기 어렵다고 겁먹지 말고, 그냥 가져다 쓰자) 다음과 같은 cost function으로 나타낼 수 있다.

cost = Sum(-y*log(sigmoid(Wx+b)) - (1-y)*log(1-(sigmoid(Wx+b)))) / n

사실 이 개념은, (0,1)의 값으로 나타낼 수 있는 데이터의 사후분포를 모델이 예측하는 사후 분포와 가장 비슷하도록 조정하는 통계적 방법인 최대 우도법(maximum likelihood estimation)으로부터 도출된 수식이다. 최대우도법을 이용한 통계적 분포 추정 방식에 비용(Cost 혹은 Error)함수라는 본질적인 목적(마이너스 혹은 플러스로 error와 항등한 성질을 가지는 결과값)에 충실하기 위해 로그를 씌워놨을 뿐이다.

(4) Gradient Descent로 학습하기

그래디언트 디센트를 이용한 학습과정은 일반적인 Regression과 동일하다. 학습의 각 단계(iteration)에서 다음의 계산을 업데이트 하면 된다. 위에서 구한 cost의 편미분값을 계산해야 하는데, 역시나 수식적인 증명은 생략하겠다.

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

3. How to run

I made my code using 'jupyter notebook'. Therefore, you can check it using 'Jupyter notebook' or 'colaboratory'.

4. A set of snapshots

1.1 라이브러리 호출

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.2 데이터 불러오기

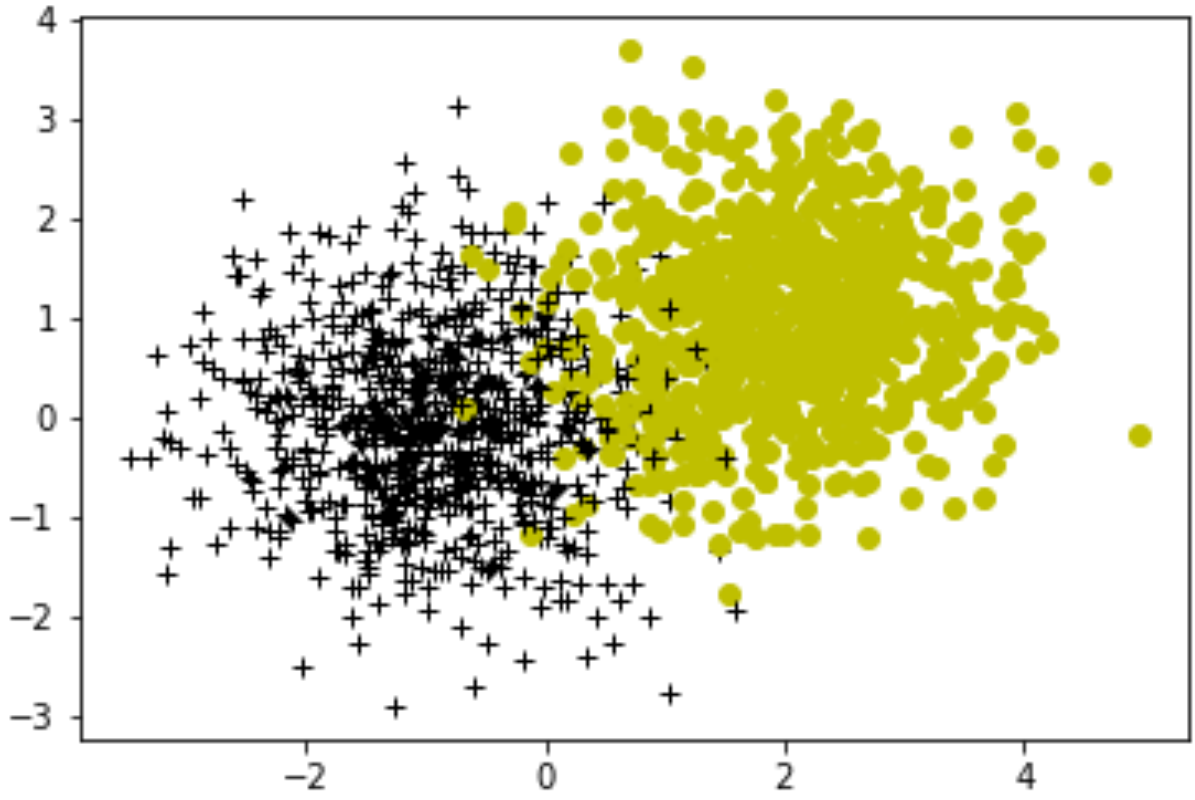
```
df = pd.read_csv('hw2_train.csv')
x1 = df['x1']
x2 = df['x2']
y = df['y']

print(df)
```

```
            x1        x2  y
0     2.948287 -0.021679  0
1    -0.535186  0.004075  1
2    -1.773161  1.033747  1
3     0.170943 -1.337417  1
4    -1.376378  0.364972  1
...        ...       ... ..
1495 -3.194169 -0.197228  1
1496  2.553774 -0.148602  0
1497 -0.794583  0.392619  1
1498 -0.485204  0.416609  1
1499  2.306127  1.257547  0

[1500 rows x 3 columns]
```

1.3 함수

- 'Hw1_predic' 함수 = 'Hw2_predic' 함수
- 색의 진함의 차이

```
def Draw():
  for i in range(1500):
    if (y[i] == 1):
      plt.plot(x1[i], x2[i], 'k+')
    elif (y[i] == 0):
      plt.plot(x1[i], x2[i], 'ko', color='y')
  plt.show()

def Draw2():
  for i in range(1500):
    if (y[i] == 1):
      plt.plot(x1[i], x2[i], 'k+')
    elif (y[i] == 0):
      plt.plot(x1[i], x2[i], 'ko', color='y')

Draw()

def Hw2_Predic(x2_pred, y):
  Draw2()
  plt.plot(x1, x2_pred, 'ko', alpha=0.1, color='blue')
  plt.show()

def Hw1_Predic(x2_pred, y):
  Draw2()
  plt.plot(x1, x2_pred, 'ko', alpha=0.7, color='blue')
  plt.show()

# 시그모이드 함수
def Sigmoid(z):
    return 1/(1+np.e**(-z))
```

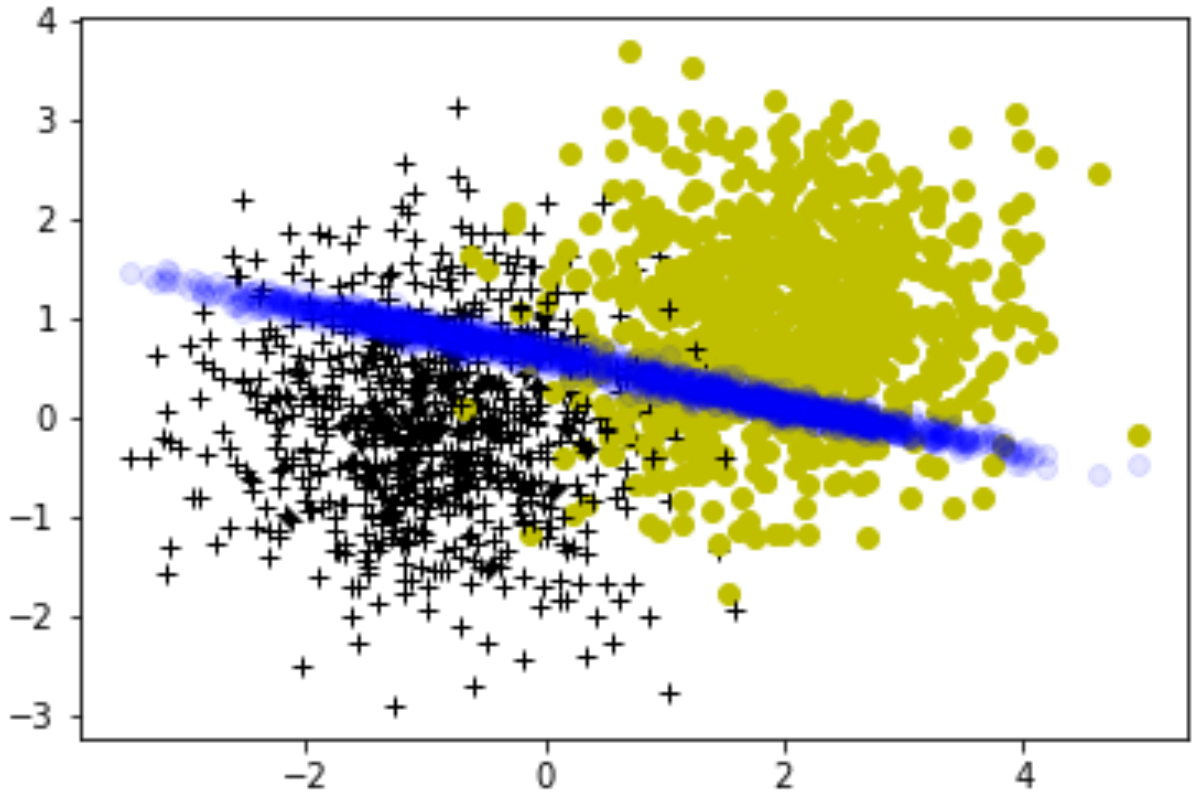


2 Task1

2.1 theta 구하기

```
# e의 지수 승 : z 구하기
A = pd.DataFrame({'1': 1, '2': df['x1'], '3': (df['x2'])})

b = pd.DataFrame({'1': df['y']})

pre_theta1 = (A.T).dot(A)                              # 괄호 안 계산

pre_theta1_reverse = np.linalg.inv(pre_theta1)         # 역행렬

pre_theta2 = A.T                                       # A의 전치행렬

pre_theta3 = b                                         # b

temp_theta = (pre_theta1_reverse).dot(pre_theta2)      # 역행렬한 것과 A의 전치행렬 곱셈

theta = temp_theta.dot(b)                              # 위의 결과와 행렬 b 곱셈

print(theta)

a = theta[2]
b = theta[1]
c = theta[0]

print('0 = ',a,'* x2 +',b,' * x1 +',c)

z = a*x2+b*x1+c

Hw2_Predic(z, x2)
```

```
[[ 0.65052637]
 [-0.22813649]
 [-0.07331323]]
0 =  [-0.07331323] * x2 + [-0.22813649]  * x1 + [0.65052637]
```



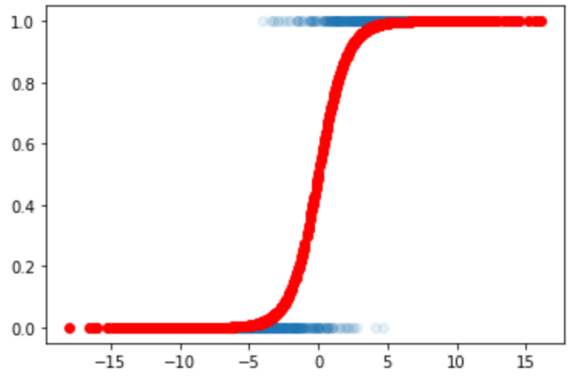
2.2 최적의 a, b, c 값 구하기

```
learning_rate = 0.01

for epoch in range(100000):
  z = a*x2+b*x1+c

  a_grad = sum(x2*((Sigmoid(z))-y))/1500
  b_grad = sum(x1*((Sigmoid(z))-y))/1500
  c_grad = sum((Sigmoid(z))-y)/1500

  a = a - learning_rate*a_grad
  b = b - learning_rate*b_grad
  c = c - learning_rate*c_grad

  if epoch%10000 == 0:
    z = a*x2+b*x1+c

    x2_pred = x1*(-(b/a))-(c/a)
    Hw1_Predic(x2_pred,x2)                    # train data + linear decision boundary 시각화
    print('\n a: ',a, 'b :', b, 'c: ',c)     # 주기를 1000으로하여 최적의 a, b, c값 출력
    print('-----------------------------------------------------')
```

```
a = -1.15073793
b = -3.77371361
c = 2.28758314

# row 생략 없이 출력
pd.set_option('display.max_rows', None)
# col 생략 없이 출력
pd.set_option('display.max_columns', None)
```

2.3 크로스 엔트로피 구하기

```
z = a*x2 + b*x1 + c
plt.plot(z, Sigmoid(z), 'ko', color='r')
plt.scatter(z, y, alpha=0.1)
plt.show()


# task1의 total cost
task1_total_cost = sum(y*np.log(Sigmoid(z))+(1-y)*np.log(1-Sigmoid(z)))/(-1500)
print("\n task1's total cost: ",task1_total_cost)
```




```
task1's total cost:  0.10636790396670684
```

2.4 Result

```
print("\n task1's a :", a)
print("\n task1's b :", b)
print("\n task1's c :", c)
print("\n task1's total cost :",task1_total_cost)
```

```
task1's a : -1.15073793

task1's b : -3.77371361

task1's c : 2.28758314

task1's total cost : 0.10636790396670684
```

3 Task2

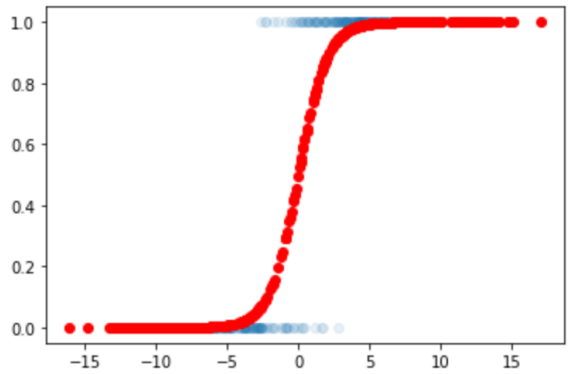
3.1 test 데이터로 test하기

```
df2 = pd.read_csv('hw2_test.csv')
```

```
df2_y = df2['y']

z2 = a*df2['x2']+b*df2['x1']+c
```

3.2 test : 시그모이드 함수 출력

```
plt.plot(z2, Sigmoid(z2), 'ko', color='r')
plt.scatter(z2, df2_y, alpha=0.1)
plt.show()
```



3.3 task2의 total cost를 구하기

```
task2_total_cost = sum(df2_y*np.log(Sigmoid(z2))+(1-df2_y)*np.log(1-Sigmoid(z2)))/(-500)
print("\n task2's total cost: ",task2_total_cost)
```

```
 task2's total cost:  0.08808280778078409
```

3.4 최적 값 a, b, c를 test data에 적용

```
test = Sigmoid(z2)

# 시그모이드 함수 결과 값이 0.5이상이면 '1' 아니면 '0'으로 분류
for i in range(len(test)):
  if (test[i] >= 0.5):
    test[i] = 1
  else:
    test[i] = 0
```

3.5 accuracy 구하기

```
success =0
fail =0
for j in range(500):
  if (test[j] == df2_y[j]):
    success += 1
  else:
    fail += 1

print("성공횟수: ", success)
print("실패횟수: ", fail)
print("accurcy: ", success/500%100, "%")
```

```
성공횟수:  483
실패횟수:  17
accurcy:  0.966 %
```

3.6 Result

```
print("\ntask2's total cost: ",task2_total_cost)
print("accurcy: ", success/500%100, "%")
```

```
task2's total cost:  0.08808280778078409
accurcy:  0.966 %
```

5. Result

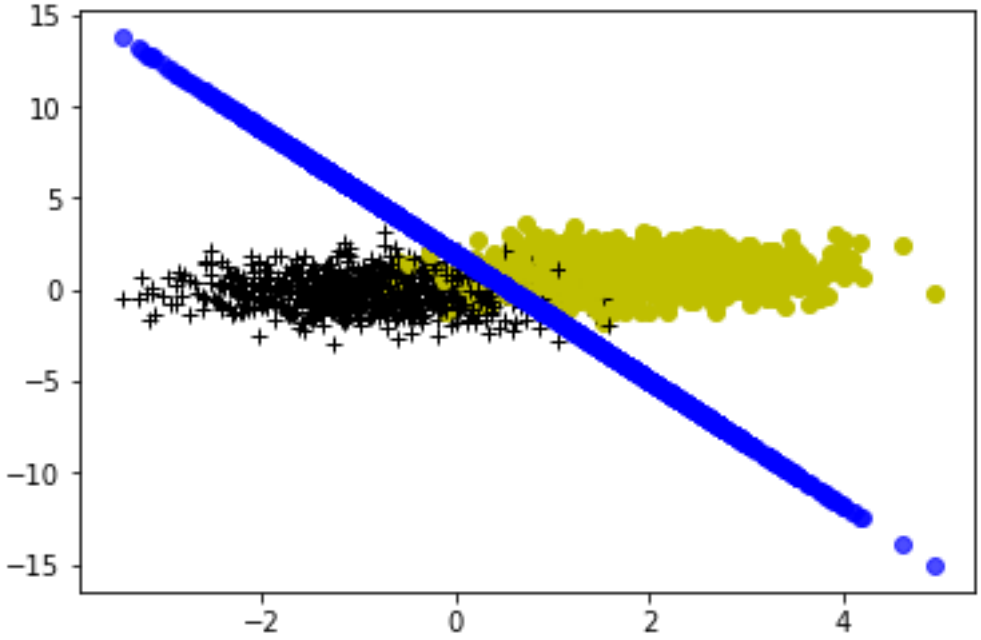
A. The parametrs of your classifier

	a	b	c
Task1	-1.15073793	-3.77371361	2.28758314

B. Total Cost for each task

	Cost
Task1	0.10636790396670684
Task2	0.08808280778078409

C. Plots of the classification results



a: [-1.15073793] b : [-3.77371361] c: [2.28758314]

D. Accuracy

Accuracy : 96.6%

6. Conclusion

예측 변수를 사용하여 이항 결과를 모형화하는 데 사용되는 통계적 방법

- 선형 회귀 분석의 개념을 결과 변수가 범주형인 상황으로 확장합니다.

예측 변수를 사용하여 이항 결과를 모형화하는 데 사용되는 통계적 방법

- 이것은 분류를 위한 것이지 회귀를 위한 것이 아니다.