Visual Computing Project No.2

여러가지 선형 분류 알고리즘을 이용하여 2D 패턴 분석 및 결과

국민대학교 컴퓨터공학과
20113337 최영근
2014년 04월 04일

차 례

시작글

1. 프로젝트 배경과 목적

여러가지 선형 분류 알고리즘을 학습하여 2D 패턴을 분류하는 프로그램을 작성하고 각각의 결과를 분석하고 차이점을 비교한다. 또한 해당 알고리즘에서 다양한 값으로 weight, learning rate, threshold, margin vector을 설정하고 그에 따른 영향을 분석한다.

2. 프로젝트 대상 과 범위 한정

Sample	w_1		w_2		w_3	
	x_1	x_2	x_1	x_2	x_1	x_2
1	0.1	1.1	7.1	4.2	-3.0	-2.9
2	6.8	7.1	-1.4	-4.3	0.5	8.7
3	-3.5	-4.1	4.5	0.0	2.9	2.1
4	2.0	2.7	6.3	1.6	-0.1	5.2
5	4.1	2.8	4.2	1.9	-4.0	2.2
6	3.1	5.0	1.4	-3.2	-1.3	3.7
7	-0.8	-1.3	2.4	-4.0	-3.4	6.2
8	0.9	1.2	2.5	-6.1	-4.1	3.4
9	5.0	6.4	8.4	3.7	-5.1	1.6
10	3.9	4.0	4.1	-2.2	1.9	5.1

➔ 각각 10개의 데이터로 나눠져 있는 10개의 데이터 w1, w2, w3 3개의 클래스에 대한 10개의 데이터에 대해 분류를 진행한다.

해당 프로젝트에서는 2-Dimension을 다루기 때문에 인풋값이 (0, 0) 일 때 아웃풋이 0이되는 결과를 방지하기 위해 bias 값을 1로 설정했다.

3. 프로젝트 구성과 내용

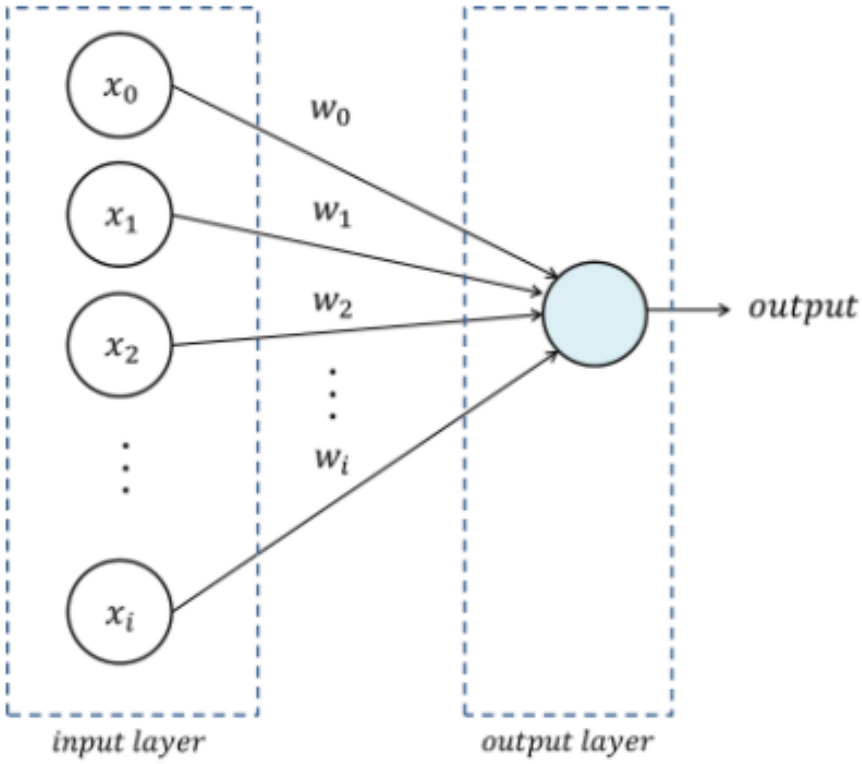
Perceptron Algorithm, Bacth relaxtion algorithm, Widrow-Hoff(LMS) algorithm 대한 간단한 설명과 해당 알고리즘으로 작성한 프로그램의 중요 부분, 최종적인 출력 결과를 보여준다.

본문

1. Perceptron Algorithm

1.1. 설명

Perceptron Algorithm 에는 여러가지 알고리즘이 있지만 프로젝트에서 요구하는 것이 w1, w2 클래스를 분류하는 선형 분류기를 작성하는 것이기 때문에 가장 간단한 Single-layer-perceptron(단층 퍼셉트론) 알고리즘을 적용하였다.



[그림 2] 출력층의 크기가 1인 단층 퍼셉트론

1.2. 프로그램 설명 및 요약

Weight값은 초기 랜덤(-0.5~0.5)으로 설정 하였고, learning rate는 0.1, 0.5, random값으로 설정하였다. 마지막으로 threshold 값은 0으로 두고 크면 1 작으면 -1 로 분류하였다.

```
"""------------------------------------------------------
    Name : perceptron()
    Input = n*3 matrix, 1*3 weight vector
    Description : input*weight -> layer -> output
                  output -> activate -> return f(net)
------------------------------------------------------"""
def perceptron(data, weight):

    temp = data[0]*weight[0] + data[1]*weight[1] + data[2]*weight[2]
    net = activateFunc(temp)

    return net
```

➔ 데이터들의 net 값을 계산하는 함수

인풋데이터에 대해 net값을 계산하고 activateFunc 함수로 들어가 threshold 값과 비교하여 클래스를 구분하게 된다.

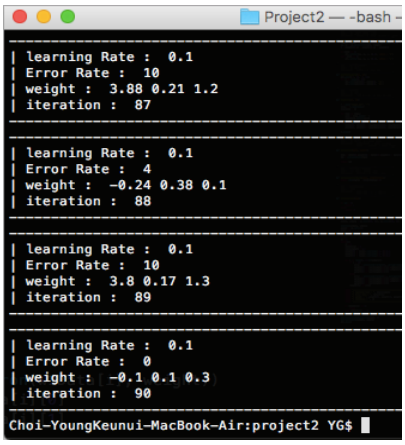
$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum_{y \in Y_k} \mathbf{y}$$

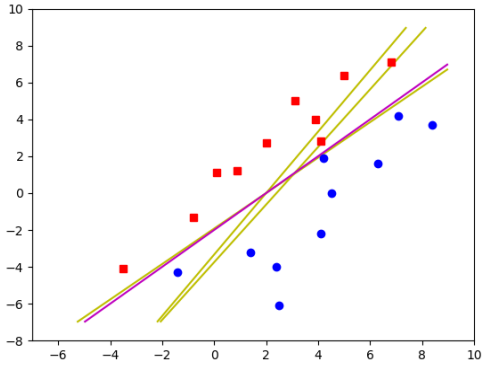
```
for i in range(0, 2*len(w1Data)):
    if i < 10:
        error = calculateErrorRate(1, perceptron(w1Data[i], weight))
        weight[0] += learningRate*error*w1Data[i][0]
        weight[1] += learningRate*error*w1Data[i][1]
        weight[2] += learningRate*error
    else:
        error = calculateErrorRate(-1, perceptron(w2Data[i-10], weight))
        weight[0] += learningRate*error*w2Data[i-10][0]
        weight[1] += learningRate*error*w2Data[i-10][1]
        weight[2] += learningRate*error
    totalError += error
```

➔ 출력층의 뉴런값과 목표값을 비교하여 다를 때 가중치를 조정하는 부분

퍼셉트론 알고리즘에 의하여 w1, w2 데이터들을 돌면서 출력값이 목표값과 다를 때 위와 같이 가중치를 조정한다. 이 알고리즘은 totalError 가 0이 될 때까지 학습을 진행한다.

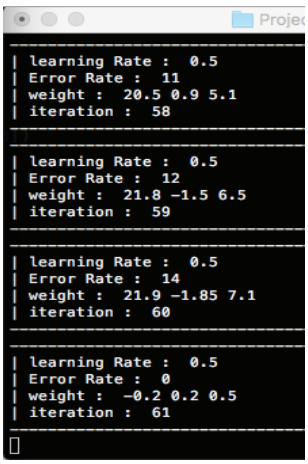
1.3. 결과

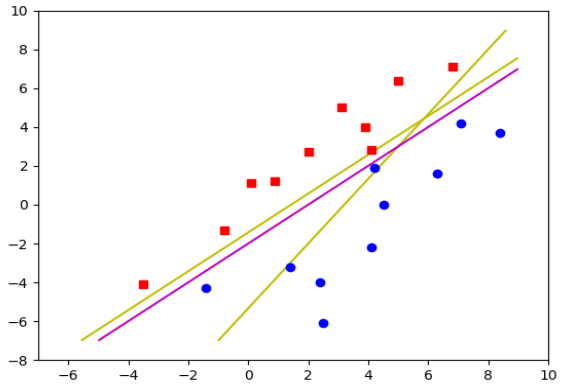




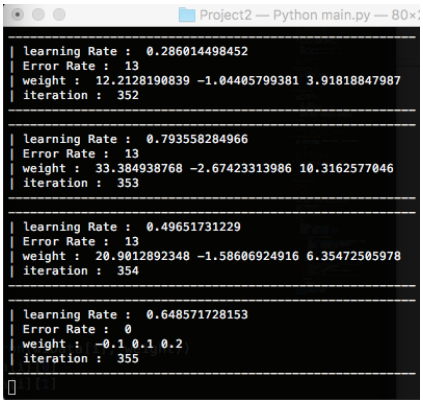
➔ learning rate 가 0.1일때 출력 및 결과

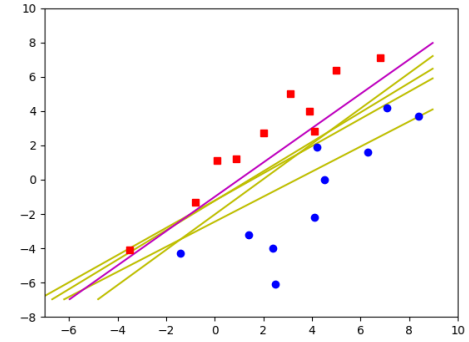
(error율이 2이하인 것만 plot하였고 자주색선이 최종 분류된 바운더리)





➔ learning rate가 0.5 일때의 출력 및 결과





➔ 새로 돌 때 마다 learning rate를 랜덤으로 초기화 했을 때 출력 및 결과

결과 : 학습율을 계속 업데이트 시키는 것 보다는 일정한 학습률을 부여하였을 때 빠른 결과를 얻을 수 있었고, 0.1보다는 0.5 일 때 조금더 빠르게 찾을 수 있었다.

2. Batch relaxation Algorithm

2.1. 설명

Relaxation algorithm 은 미분 방정식의 유한 차분 이산화로
발생한 거대하고 희소 한 선형 시스템 을 해결하기 위해 개발된 알고리즘이다.
Perceptron 알고리즘과 유사하지만 가중치를 조정하는 과정이 조금 더 이완된 공식을 사용한다.

- The gradient of J_r is given by

$$\nabla J_r = \sum_{y \in Y} \frac{\mathbf{a}^t\mathbf{y} - b}{\| \mathbf{y} \|^2}\mathbf{y}$$

- Update Rule:

$$\begin{cases} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum \frac{b - \mathbf{a}^t\mathbf{y}}{\| \mathbf{y} \|^2}\mathbf{y} \end{cases}$$

➔ 가중치 업데이트 규칙 공식

2.2. 프로그램 설명 및 요약

Weight값은 초기(0, 0, 0)으로 설정 하였고, learning rate는 0.1, 0.5, random값으로 설정하였다. threshold 값은 perceptron 과 동일하게 0으로 설정하였고, margin vector을 0.1과 0.5로 설정하여 진행하였다.

```
for i in range(0, 2*len(w1Data)):
    if i < 10:
        tempY = w1Data[i][0]*w1Data[i][0]+w1Data[i][1]*w1Data[i][1]+1
        temp = (margin - np.dot(weight, w1Data[i]))/tempY
        if calculateErrorRate(-1, perceptron(w1Data[i], weight)) == 1:
            weightSum[0] += learningRate*temp*w1Data[i][0]
            weightSum[1] += learningRate*temp*w1Data[i][1]
            weightSum[2] += learningRate*temp*w1Data[i][2]
            totalError +=1

    else:
        tempY = w3Data[i-10][0]*w3Data[i-10][0]+w3Data[i-10][1]*w3Data[i-10][1]+1
        temp = (margin - np.dot(weight, w3Data[i-10]))/tempY
        if calculateErrorRate(1, perceptron(w3Data[i-10], weight)) == 1:
            weightSum[0] += learningRate*temp*w3Data[i-10][0]
            weightSum[1] += learningRate*temp*w3Data[i-10][1]
            weightSum[2] += learningRate*temp*w3Data[i-10][2]
            totalError +=1

weight[0] += weightSum[0]
weight[1] += weightSum[1]
weight[2] += weightSum[2]
```

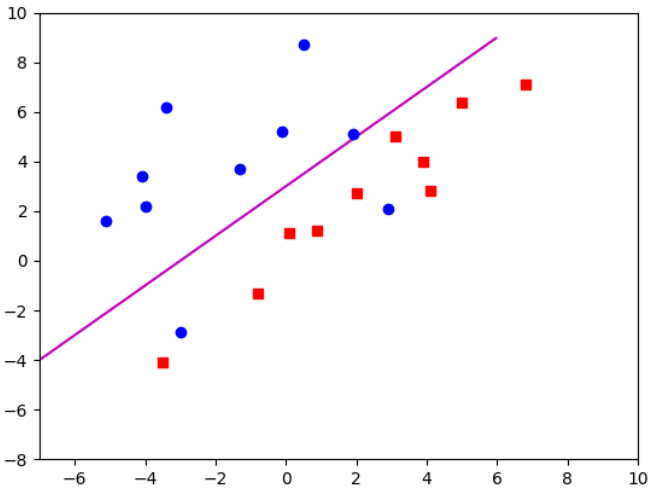
➔ relaxation algorithm에 해당하는 공식을 적용한 코드

초기 (0,0,0)으로 초기화된 가중치로 학습을 진행하고 잘못 분류된 데이터가 나타나면 가중치 업데이트 규칙에 의해 가중치를 증가시켜 주었고, 바로바로 업데이트 시키지 않고 일괄적으로 맨 마지막에 업데이트 하는 방식을 적용하였다. 1번의 퍼셉트론 알고리즘을 적용했을 때와 마찬가지로 total error rate을 계산하면서 진행하였다.

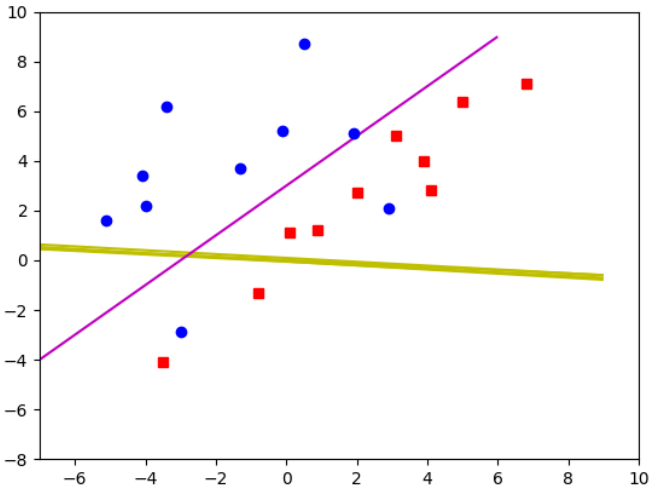
2.3. 결과

w1 과 w3의 데이터는 선형으로는 분류가 불가능한 형태의 구조를 띄기 때문에 Total Error가 0이 되는 부분을 종료조건으로 두게되면 무한루프에 빠지는 것을 찾았고, 랜덤한 가중치로 계속 실험한 결과 어느 정도 두 데이터를 최선으로 나눌 수 있는 바운더리를 구할 수는 있었다. 또한 1번과 마찬가지로 learning rate는 0.1 보다는 0.5 가 조금 더 빠르게 수렴하였고, margin b 값도 0.1 보다는 0.5로 했을 때

조금 더 빠르게 수렴하였다. 그러나 학습률을 너무 크게 주면 발산하는 것을 볼 수 있었다.



➔ learning rate = 0.5 일 때의 출력 결과 (500번 돌림)

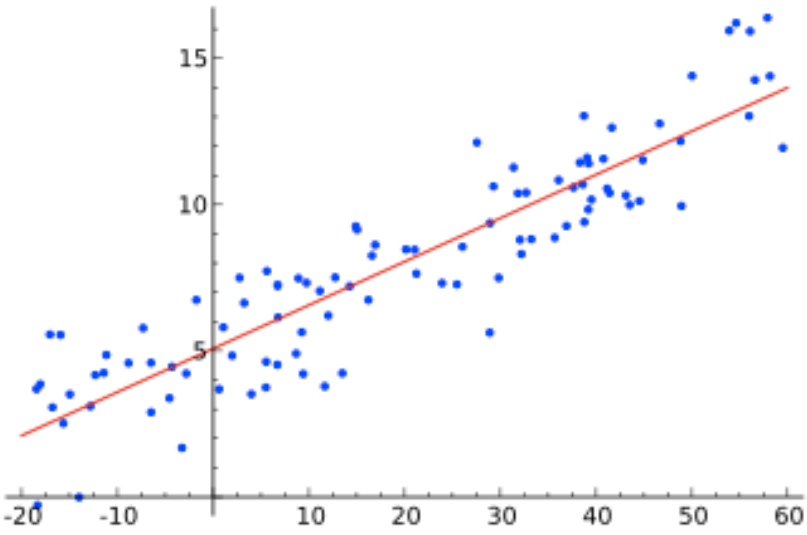


➔ learning rate = 0.1 일때의 출력 결과 (500번 돌림)

3. Widrow-Hoff Procedure (LMS)

3.1. 설명

이 방법은 많은 데이터가 주어졌을 때 그 데이터를 이루는 그래프를 산출해 내기위한 방법 중 하나이다. 입력된 데이터에 대해 오차가 가장 적은 값을 선택한다. 아주 간단한 근사화 방법으로 실제로 많이 쓰이지만 정확도가 높지 않다는 단점을 가진다.



➔ 파란색 점들의 경향성을 따라 LMS로 직선을 그림

$$\begin{cases} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) & = \mathbf{a}(k)+\eta(k)(\mathbf{b}_k-\mathbf{a}^t(k)\mathbf{y}^k)\mathbf{y}^k \end{cases}$$

➔ LMS 의 가중치 업데이트 공식

3.2. 프로그램 설명 및 요약

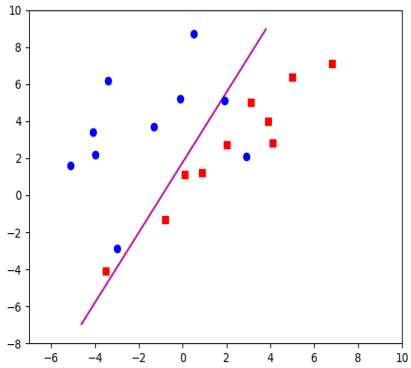
LMS 알고리즘으로 w1, w3을 구분하는 선형 분류기를 작성한다. 다양한 weight vector, learning rate, margin b, threshold를 주어 결과를 분석한다. Weight vector 는 다양한 값을 주기 위해 random한 값으로 받았고, learning rate와 margin b 값은 리스트의 형태로 만들어 매번 랜덤 값으로 실험하였고, threshold는 0과 1로 실험하였다.

```
for i in range(0, 2*len(w1Data)):
    if i < 10:
        temp = w1Data[i][0]*weight[0] + w1Data[i][1]*weight[1] + w1Data[i][2]*weight[2]
        error = calculateErrorRate(1, perceptron(w1Data[i], weight, th))
        weight[0] += learningRate*error*(margin[i]-temp)*w1Data[i][0]
        weight[1] += learningRate*error*(margin[i]-temp)*w1Data[i][1]
        weight[2] += learningRate*error*(margin[i]-temp)
    else:
        temp = w3Data[i-10][0]*weight[0] + w3Data[i-10][1]*weight[1] + w3Data[i-10][2]*weight[2]
        error = calculateErrorRate(-1, perceptron(w3Data[i-10], weight, th))
        weight[0] += learningRate*error*(margin[i-10]-temp)*w3Data[i-10][0]
        weight[1] += learningRate*error*(margin[i-10]-temp)*w3Data[i-10][1]
        weight[2] += learningRate*error*(margin[i-10]-temp)
    totalError += error
```

➔ lms 알고리즘을 적용하여 작성한 코드

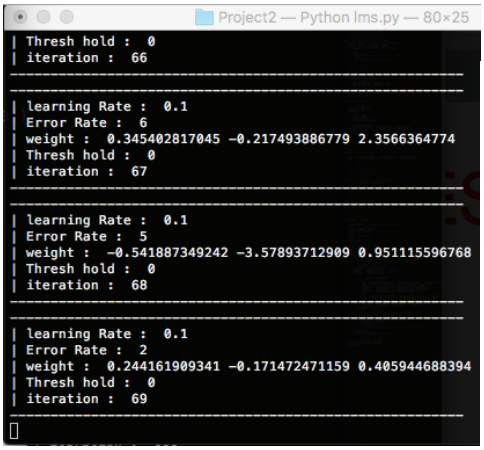
3.3. 결론

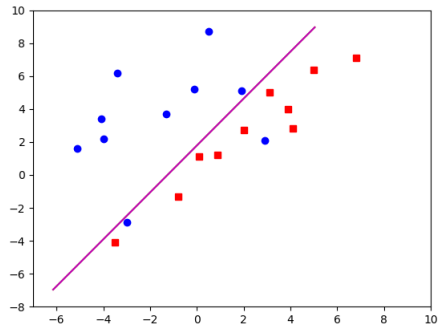
```
Project2 — Python lms.py — 80×25
| Thresh hold :  0
| iteration :  53
------------------------------------------------------------
------------------------------------------------------------
| learning Rate :  0.260650291577
| Error Rate :  4
| weight :  -0.878874683196 -48.2165808753 -1.3416615627
| Thresh hold :  0
| iteration :  54
------------------------------------------------------------
------------------------------------------------------------
| learning Rate :  0.0485237012364
| Error Rate :  6
| weight :  -0.0658228391842 -0.837613878795 -0.128632412274
| Thresh hold :  0
| iteration :  55
------------------------------------------------------------
------------------------------------------------------------
| learning Rate :  0.322215048538
| Error Rate :  1
| weight :  0.333169984914 -0.175980355752 0.411437925833
| Thresh hold :  0
| iteration :  56
------------------------------------------------------------
```



➔ threshold = 0, learning rate = random 의 결과

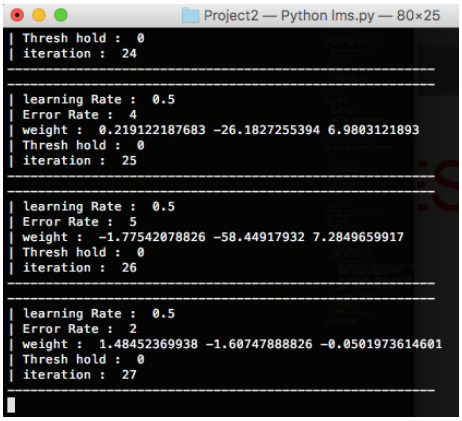
threshold = 0, learning rate 를 매번 랜덤하게 부여하였을 때 56 번만에 두 클래스를 분류하는 가장 좋은 바운더리를 얻을 수 있었다.

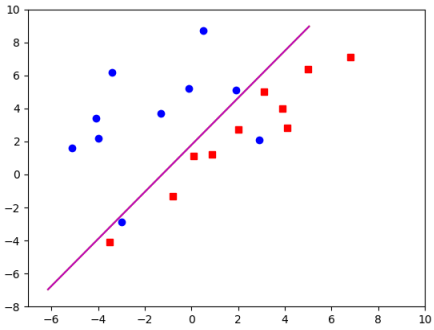




➔ learning rate = 0.1, threshold = 0 일 때의 결과

반복 69번 만에 2개의 에러를 가지는 선형 분류기를 생성하였음을 알 수 있다.





➔ learning rate = 0.5, threshold = 0 일 때의 결과

반복 27번 만에 2개의 에러를 가지는 선형 분류기를 생성하였음을 알 수 있다.

따라서 1, 2번과 마찬가지로 learning rate가 0.1 일 때 보다 0.5일 때 조금 더 빨리 수렴하는 것을 찾을 수 있었다. 그리고 threshold 값은 0이 아닌 다른 값을 주었을 때 조금 더 부정확한 결과를 도출하였다.

결론