فاطمه نائینیان **810198479** **تمرین 3 ماشین لرنینگ**

1)

(1

$$P(x) = \frac{1}{nh_n} \sum_{i=1}^{n} \varphi\left(\frac{x-x_i}{h_n}\right) \quad \begin{cases} \varphi(x) \sim N(0,1) \\ P(x) \sim N(\mu, \sigma^2) \end{cases}$$

① $$E[P(x)] = E\left[\frac{1}{nh_n} \sum_{i=1}^{n} \varphi\left(\frac{x-x_i}{h_n}\right)\right] = \frac{1}{nh_n} \sum_{i=1}^{n} E\left[\varphi\left(\frac{x-x_i}{h_n}\right)\right] = \frac{1}{nh_n} \sum_{i=1}^{n} \varphi\left(\frac{x-x_i}{h_n}\right) P(x) Q_n$$

$$\bar{P}_n(x) = \sum \varphi\left(\frac{x-x_i}{h_n}\right) P(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2h_n^2}\right) \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

$$= \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2(\sigma^2+h_n^2)}\right) \sim N(\mu, h_n^2+\sigma^2)$$

② $$P(x) - \bar{P}_n(x) = \frac{1}{\sqrt{2\pi}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] - \frac{1}{\sqrt{2\pi}\sqrt{h_n^2+\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2(h_n^2+\sigma^2)}\right]$$

$$= \underbrace{\frac{1}{\sqrt{2\pi}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]}_{P(x)} \left(1 - \underbrace{\frac{1}{\sqrt{h_n^2+\sigma^2}}} \underbrace{\exp\left[\frac{-(x-\mu)^2}{2(h_n^2+\sigma^2)}\right]}\right)$$

برای h_n کوچک $\frac{1}{\sigma}\left[1+\frac{h_n^2}{2\sigma^2}\right]$ برای h_n کوچک $1-\frac{(x-\mu)^2}{2(h_n^2+\sigma^2)}$

$$\rightarrow P_n(x) - \bar{P}_n(x) = \left[1 - \frac{1}{\sigma}\left[1+\frac{h_n^2}{2\sigma^2}\right]\left[1-\frac{(x-\mu)^2}{2(h_n^2+\sigma^2)}\right]\right] P(x)$$

$$= \left[1 - \frac{1}{\sigma}\left(1 + \frac{h_n^2}{2\sigma^2} - \frac{(x-\mu)^2}{2(h_n^2+\sigma^2)} - \frac{h_n^2(x-\mu)^2}{4\sigma^2(h_n^2+\sigma^2)}\right)\right] P(x)$$

چون h_n کوچک است پس می توان در برابر ۱ از آن صرف نظر کرد

$$\rightarrow P_n(x) - \bar{P}_n(x) = \frac{1}{2}\left(\frac{h_n}{\sigma}\right)^2 \left[1 - \left(\frac{x-\mu}{\sigma}\right)^2\right] P(x)$$

③ $$\mathrm{var}(P_n(x)) = \mathrm{Var}\left(\frac{1}{nh_n} \sum \varphi\left(\frac{x-x_i}{h_n}\right)\right) = \frac{1}{n^2h_n^2}\left[E\left(\varphi\left(\frac{x-\mu}{h_n}\right)^2\right) - E\left(\varphi\left(\frac{x-\mu}{h_n}\right)\right)^2\right]$$

$$= \frac{1}{n^2h_n^2}\left[\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{h_n^2}\right) \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) dx - \left[\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2h_n^2}\right) \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) dx\right]^2\right]$$

$$= \frac{1}{n^2h_n^2}\left[\underbrace{\int_{-\infty}^{\infty} \exp\left(\frac{-(x-\mu)^2}{h_n^2}\right) P_n(x)\, dx}_{\text{برای } h_n \text{ کوچک}:\ \frac{1}{2\sqrt{\pi}} nh_n P_n(x)} - \underbrace{\frac{h_n^2}{\sqrt{h_n^2+\sigma^2}} \times \frac{1}{\sqrt{2\pi}} \exp\left[\frac{-(x-\mu)^2}{2(h_n^2+\sigma^2)}\right]}_{\text{برای } h_n \text{ کوچک} \approx 0}\right]$$

$$\mathrm{Var}(P_n(x)) = \frac{1}{n^2h_n^2} \times \frac{1}{2\sqrt{\pi}} nh_n P_n(x) = \frac{P_n(x)}{2nh_n\sqrt{\pi}}$$

(2

$$D(a,b)=\sqrt{\sum_{k=1}^{l}(a_k-b_k)^2} \rightsquigarrow x'_k = a_k\, x_k$$

۲

① فاصله باید نامنفی باشد

$$D'(x,y)=\sqrt{\sum_{k=1}^{l}(x'_k-y'_k)^2}\geq 0 \Rightarrow (x'_k-y'_k)^2\geq 0 \Rightarrow (a'x_k-b'x_k)^2\geq 0$$

رابطه بالا همیشه برقرار است زیرا اگر که تفاضل را به توان ۲ برسانیم همیشه مثبت می‌شود.

② فاصله x از y باید با فاصله y و x برابر باشد

$$D'(x,y)=\sqrt{\sum_{k=1}^{l}(x'_k-y'_k)^2}=\sqrt{\sum_{k=1}^{l}(y'_k-x'_k)^2}=D'(y,x)$$

③ مجموع فاصله x از y و y و z بزرگتر مساوی فاصله x و z باشد

$$D'(x,y)+D'(y,z)\geq D'(x,z)$$

فرض کنیم دیتاها تک بعدی باشند

$$D'(x,y)=(x'_k-y'_k)^2 \qquad D'(y,z)=(y'_k-z'_k)^2 \qquad D(x,z)=(x'_k-z'_k)^2$$

$$(x'_k-y'_k)^2+(y'_k-z'_k)^2\geq(x'_k-z'_k)^2$$

$$x'^2_k-2x'_ky'_k+y'^2_k+y'^2_k-2y'_kz'_k+z'^2_k\geq x'^2_k-2x'_kz'_k+z'^2_k$$

$$2y'_k(y'_k-x'_k-z'_k)\geq -2x'_kz'_k \rightsquigarrow y'^2_k+x'_kz'_k-y'_kx'_k-y'_kz'_k\geq 0$$

$$y'_k(y'_k-x'_k)-z'_k(y'_k-x'_k)\geq 0 \rightsquigarrow (y'_k-z'_k)(y'_k-x'_k)\geq 0$$

این رابطه همیشه برقرار است

در knn از این موضوع استفاده می‌کنند تا حجم محاسبات را کم کنند.

(3

(۳)

الف) خطا در حالتی رخ می دهد که x عضو کلاس w_1 باشد ولی label آن w_2 نشان داده شود و به عکس. پس:

$$P(error) = P(x \in D_1 | w_2) + P(x \in D_2 | w_1)$$

چون هر دو توزیع یکنواخت دارند و مرکز آنها ۱ واحد فاصله دارد پس احتمال خطای ①,② برابر است

$$P(error) = 2P(x \in D_1 | w_2)$$

حال فرض می‌کنیم در هر مرحله j داده label اشتباه بخورند یعنی

$$P(error) = \frac{1}{2^Q} \sum_{j=0}^{\frac{k-1}{2}} \binom{Q}{j}$$

ب) در حالتی که سلیما دادیم یعنی احتمال خطاها جمع می‌شود پس بهترین حالت وقتی است که کمترین تعداد خطاها داشته باشیم پس اگر $k \leq 2$ باشد یعنی برای مثال $k=1$ احتمال خطا برابر $P(error) = \frac{1}{2^Q}$ می‌شود که مینیمم خطاست

ج)

$$\lim_{Q \to \infty} P(error) = \lim_{Q \to \infty} \frac{1}{2^Q} \sum_{j=0}^{\frac{k-1}{2}} \binom{Q}{j} = 0$$

سرعت رشد 2^Q خیلی بیشتر از $\sum_{j=0}^{\frac{k-1}{2}} \binom{Q}{j}$ است.

(4

٤ بخش اول

— پارامتر hn در پارزن نشان دهنده میزان sharp یا smooth بودن است به طوری که اگر hn کم باشد منحنی smooth تر می‌شود و هرچه hn بزرگتر باشد منحنی sharp تر می‌شود
برای hn کم واریانس کم و بایاس زیاد است و در hn زیاد واریانس زیاد و بایاس کم است.

— در روش پارامتریک با کمک دیتاها یک توزیع تخمین می‌زنیم که بعد از تخمین توزیع دیگر نیازی به دیتاها نداریم ولی در روش نان پارامتریک در هر مرحله به تمام داده‌ها نیاز داریم و در مقایسه این دو روش می‌توان گفت روش پارامتریک کم هزینه‌تر است و حافظه کمتری نیاز دارد ولی در روش نان پارامتریک دقت بیشتر است و به حافظه و هزینه بیشتری نیاز است.

— همانطور که گفته شد مشکل روش های kernel based این است که همیشه به دیتاها نیاز است و به دیتاهای زیادی نیاز دارد و سبب افزایش هزینه، افزایش حافظه مورد نظر و کند شدن سرعت می‌شود.

— در روش پارزن با ثابت نگه داشتن حجم تعداد دیتاهای موجود در حجم را اندازه می‌گیریم و در روش knn با ثابت کردن تعداد دیتا، حجم را محاسبه می‌کنیم

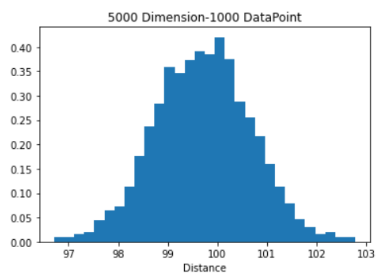
بخش دوم

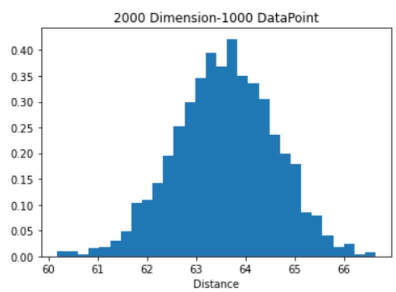
2) در روش knn، تعداد نقاط یعنی k ثابت است و Vn یا همان حجم تغییر می‌کند حال درصورتی که بعد را ثابت فرض کنیم و تعداد داده‌ها را افزایش دهیم علاوه بر افزایش حجم محاسبات، پیچیدگی محاسبه فاصله نیز افزایش می‌یابد و این افزایش (زمانی که صرفاً بعد افزایش یابد) خیلی بیشتر است
curse of dimonsionality : افزایش بعد سبب افزایش محاسبات می‌شود و باید n یعنی تعداد داده‌ها هم با همان سرعت افزایش یابد

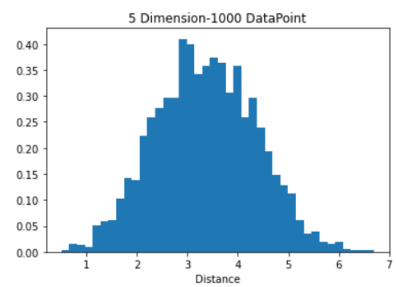
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist
```

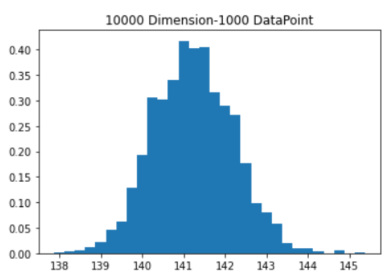
```python
def make_data(N, d , rseed=1):
    rand = np.random.RandomState(rseed)
    x = rand.randn(N*d)
    x = np.reshape(x , (N , d))
    return x

def distance(x , N , d):
    x_distance = [];
    for i in range(N-1):
        for j in (i+1 ,N-1) :
            dist = 0
            for dim in range(d):
                dist = dist + np.power(x[i][dim]-x[j][dim],2)
            x_distance.append(np.sqrt(dist))
    return x_distance
```

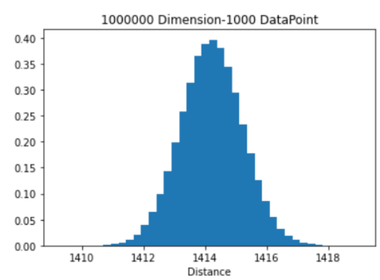
```python
x = make_data(1000 , 5)
x_distance = distance(x ,1000 , 5)
hist_distance = plt.hist(x_distance, bins=40, density=True)
plt.title("5 Dimension-1000 DataPoint")
plt.xlabel("Distance")
```

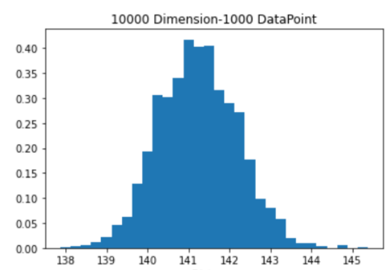


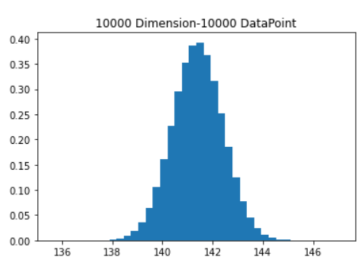


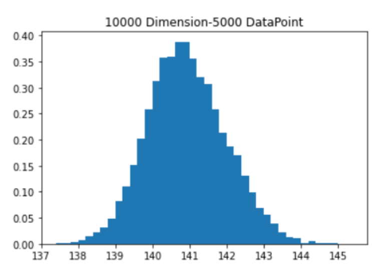


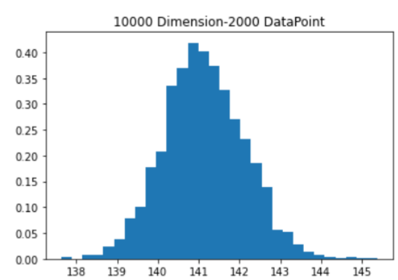












(5

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.neighbors import KernelDensity
```

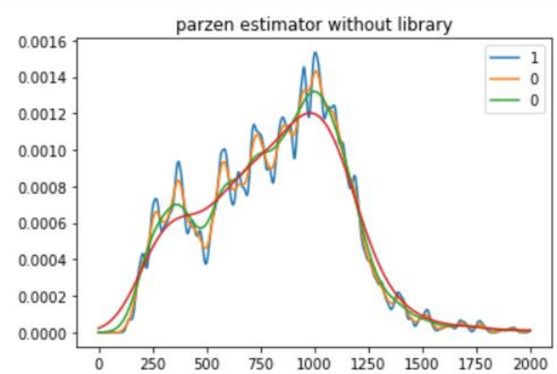
```
ted = pd.read_csv('ted_main.csv')
ted.head()
```

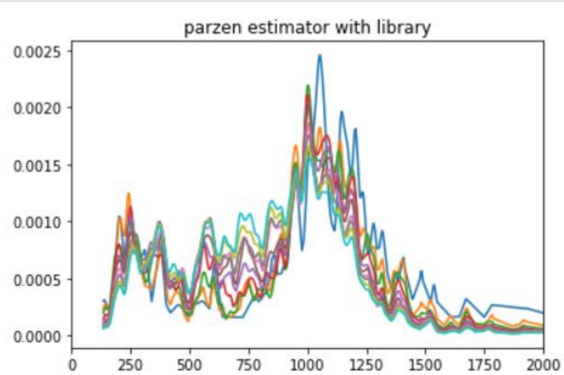
```
ted_duration = ted.duration
f, ax = plt.subplots(figsize=(7, 7))
ax.scatter(ted_duration,ted_duration,
        marker='o', color='green', s=4, alpha=0.3)
```

```
def gaussian_kernel(x):
    return (1 / (np.sqrt(2 * np.pi))) * np.exp((-0.5) * (x**2))


def parzen_window_func(x_vect, h ,x):
    parzen = []
    for x_i in x:
        k_n =0
        for j in range(len(x_vect)):
            k_n = k_n + gaussian_kernel((x_i-x_vect[j])/h)
        parzen.append(k_n/(len(x_vect)*h))
    return parzen
```

```
h = [10, 20, 50, 100]
count = np.arange(2000)
for i in h:
    estimation = parzen_window_func(ted_duration, i, count)
    plt.legend('{}'.format(i))
    plt.plot(x, estimation)
plt.title("parzen estimator without library")
plt.show()
```

```
h = [10, 20, 50, 100]
for i in range(250, len(ted_duration), 250):
    Window = KernelDensity(kernel = "gaussian", bandwidth = 10)
    Window.fit(ted_duration.reshape(-1, 1))
    plt.plot(ted_duration, np.exp(Window.score_samples(ted_duration.reshape(-1, 1))))
plt.title("parzen estimator with library")
plt.show()
```





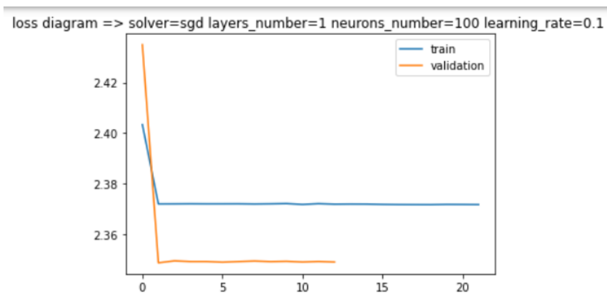
(6

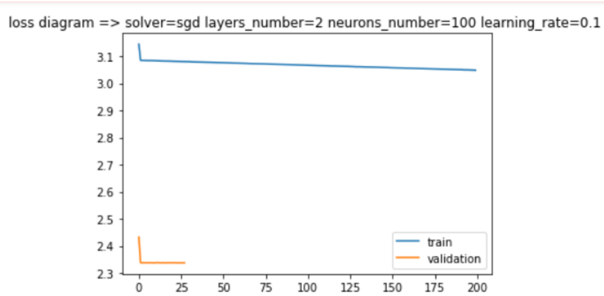
```python
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

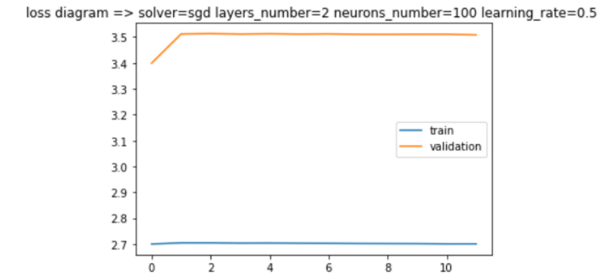
```python
def mlp(Xtrain, Xtest, Xvalid, Ytrain, Ytest, Yvalid, Solver, lr, layers, neurons):
        clf = MLPClassifier(solver = Solver, learning_rate = "constant", learning_rate_init = lr
                            , hidden_layer_sizes = (neurons, layers))
        clf.fit(Xtrain, Ytrain)
        plt.plot(clf.loss_curve_)
        clf.fit(Xvalid, Yvalid)
        plt.plot(clf.loss_curve_)
        plt.legend(["train", "validation"])
        plt.title("loss diagram => solver={} layers_number={} neurons_number={} learning_rate={} ".format(Solver,
                                                                                                   layers, neurons, lr))

        plt.show()
        Ypred = clf.predict(Xtest)
```

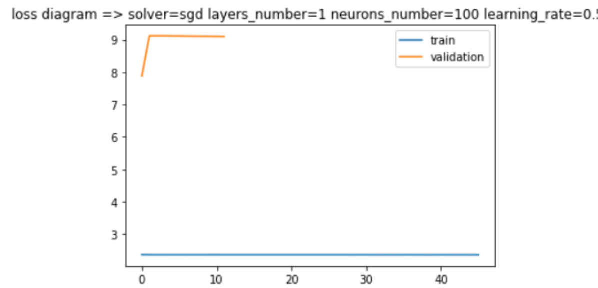
```python
train = pd.read_csv('fashion-mnist_train.csv')
y_train = train['label']
x_train = train
del x_train['label']

test = pd.read_csv('fashion-mnist_test.csv')
y_test = test['label']
x_test = test
del x_test['label']

x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size = 0.2)
```

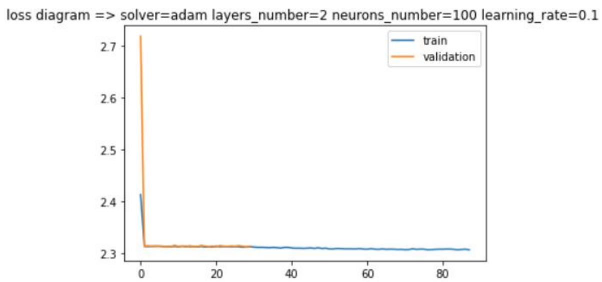
```python
layer = [1,2]
neuron = [100]
solver = ['sgd', 'adam']
learning_rate = [0.1,0.5]

for S in solver:
    for LR in learning_rate:
        for L in layer:
            for N in neuron:
                mlp(x_train, x_test, x_valid, y_train, y_test, y_valid, S, LR, L, N)
```

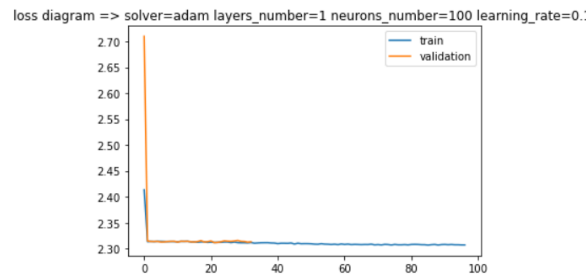


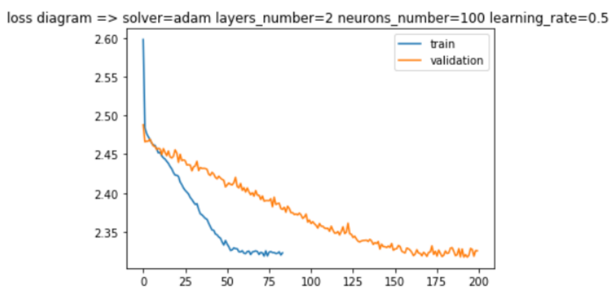


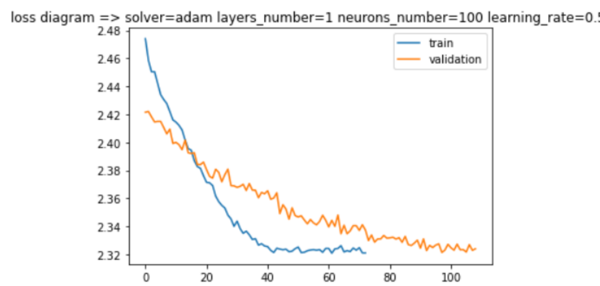

loss diagram => solver=sgd layers_number=2 neurons_number=100 learning_rate=0.5
train
validation


loss diagram => solver=sgd layers_number=1 neurons_number=100 learning_rate=0.5
train
validation


loss diagram => solver=adam layers_number=2 neurons_number=100 learning_rate=0.1
train
validation


loss diagram => solver=adam layers_number=1 neurons_number=100 learning_rate=0.1
train
validation


loss diagram => solver=adam layers_number=2 neurons_number=100 learning_rate=0.5
train
validation


loss diagram => solver=adam layers_number=1 neurons_number=100 learning_rate=0.5
train
validation

(7

```python
import matplotlib.pyplot as plt
import numpy as np
```

```python
def Perceptron(X, W, Y, LR):
    Sum = 0
    for i in range(len(W)):
        Sum += X[i] * W[i]
    if Sum > 0:
        Ypred = 1
    else:
        Ypred = 0
    if Y != Ypred:
        err = Y - Ypred
        for i in range(len(W)):
            W[i] = W[i] + (LR * err)
    return W
```

```python
learning_rate = 0.1
weight = [0.8, 0.8, 0.8]

x = [2, 2.5, 3, 0, 1, 1, 2, 2, 3, 3]
y = [0, 0, 0, 0, -1, 1, -2, 2, -3, 3]
Label = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

for i in range(50):
    for j in range(len(Label)):
        Weights = Perceptron([x[j], y[j], 1], weight , Label[j],learning_rate)
```

```
x_line = np.arange(len(x))
first_line = []
for i in x_line:
    first_line.append((i * Weights[0]) + (i * Weights[1]))
second_line = []
for i in x_line:
    second_line.append((-1 * i * Weights[0]) - (i * Weights[1]))
plt.scatter(x[:3], y[:3], c = "red")
plt.scatter(x[3:], y[3:], c = "blue")
plt.plot(x_line, first_line, c = "purple")
plt.plot(x_line, second_line, c = "purple")
plt.title("mlp")
```

```
Text(0.5, 1.0, 'mlp')
```