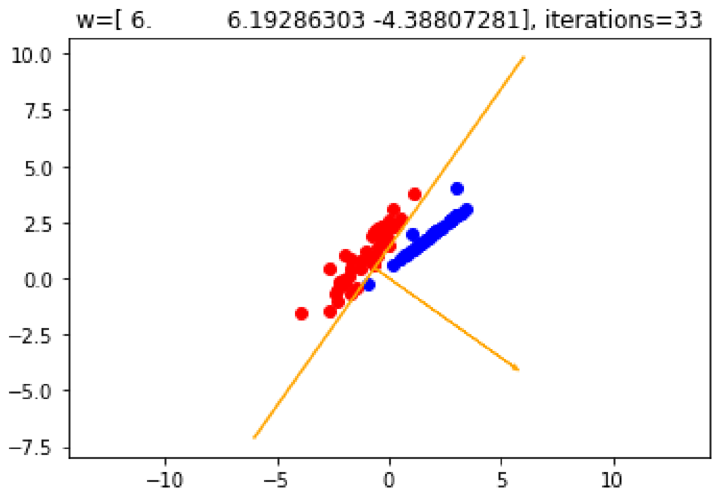
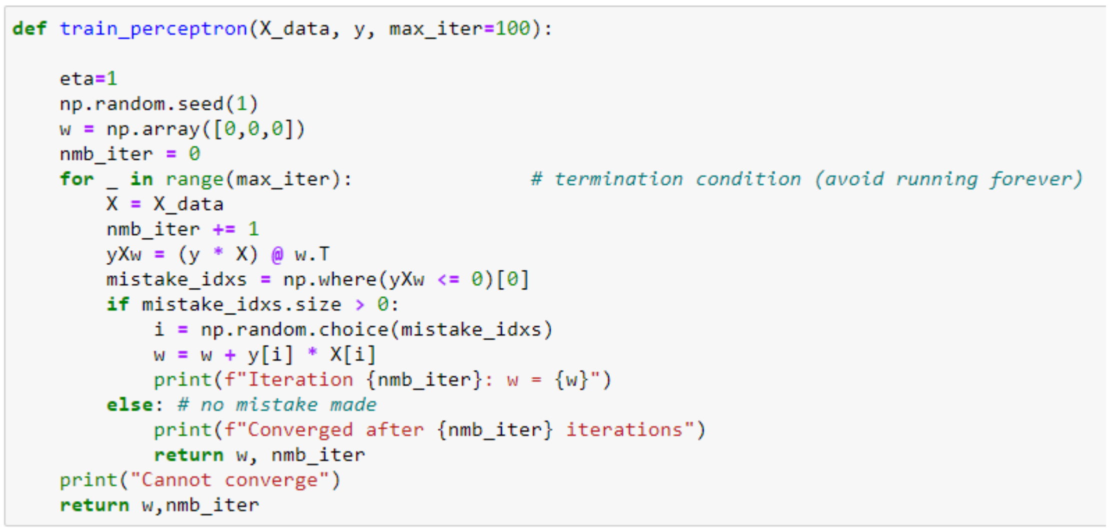
Z5305320
Dong AO
(a):

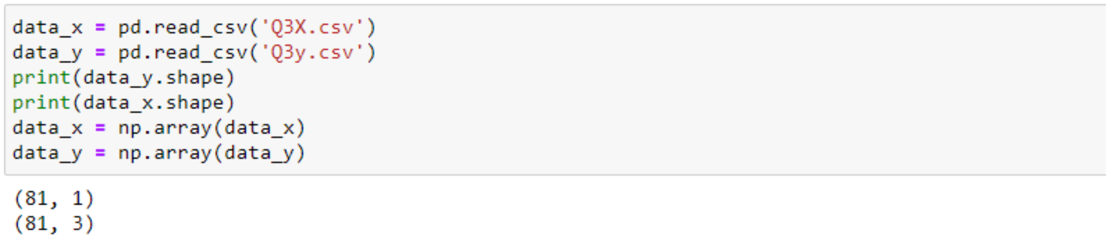


```python
import numpy as np
import pandas as pd # not really needed, only for preference
import matplotlib.pyplot as plt
```

```python
def plot_perceptron(ax, data_X, data_y, w):
#     print(np.where(y==1)[0])
    X = pd.DataFrame(data_X)
    y = pd.DataFrame(data_y)
    pos_points = X.iloc[np.where(y==1)[0]]
    neg_points = X.iloc[np.where(y==-1)[0]]
    ax.scatter(pos_points[1], pos_points[2], color='blue')
    ax.scatter(neg_points[1], neg_points[2], color='red')
    xx = np.linspace(-6,6)
    yy = -w[0]/w[2] - w[1]/w[2] * xx
    ax.plot(xx, yy, color='orange')
    ratio = (w[2]/w[1] + w[1]/w[2])
    xpt = (-1*w[0] / w[2]) * 1/ratio
    ypt = (-1*w[0] / w[1]) * 1/ratio
    ax.arrow(xpt, ypt, w[1], w[2], head_width=0.2, color='orange')
    ax.axis('equal')
```

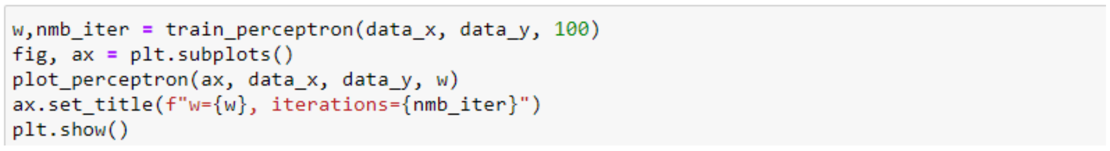


```python
def train_perceptron(X_data, y, max_iter=100):

    eta=1
    np.random.seed(1)
    w = np.array([0,0,0])
    nmb_iter = 0
    for _ in range(max_iter):                # termination condition (avoid running forever)
        X = X_data
        nmb_iter += 1
        yXw = (y * X) @ w.T
        mistake_idxs = np.where(yXw <= 0)[0]
        if mistake_idxs.size > 0:
            i = np.random.choice(mistake_idxs)
            w = w + y[i] * X[i]
            print(f"Iteration {nmb_iter}: w = {w}")
        else: # no mistake made
            print(f"Converged after {nmb_iter} iterations")
            return w, nmb_iter
    print("Cannot converge")
    return w,nmb_iter
```



```python
data_x = pd.read_csv('Q3X.csv')
data_y = pd.read_csv('Q3y.csv')
print(data_y.shape)
print(data_x.shape)
data_x = np.array(data_x)
data_y = np.array(data_y)
```

```
(81, 1)
(81, 3)
```

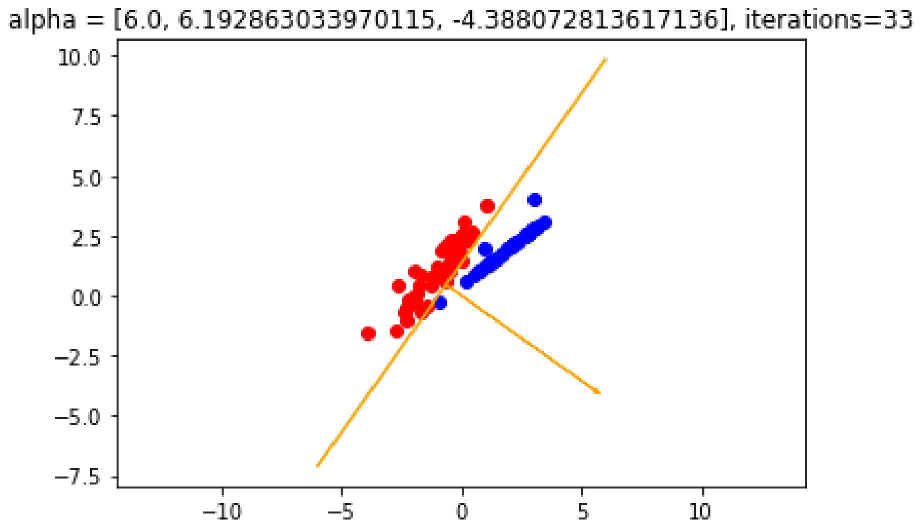


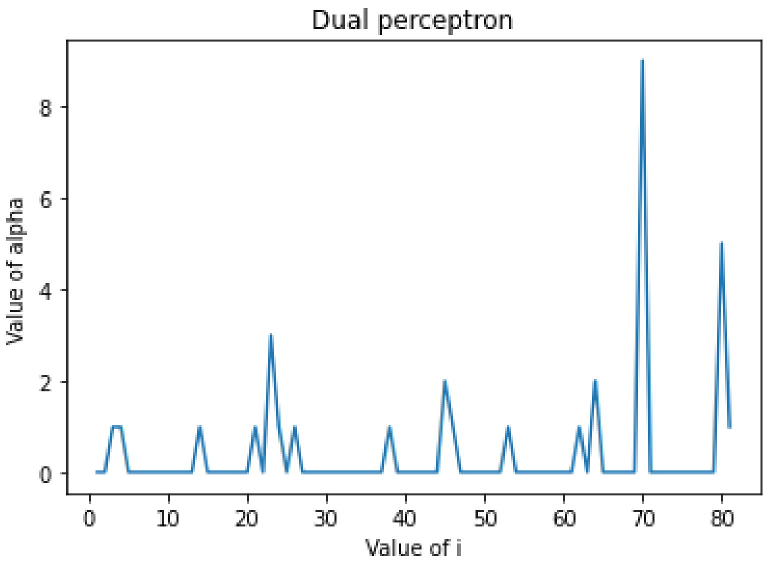
```python
w,nmb_iter = train_perceptron(data_x, data_y, 100)
fig, ax = plt.subplots()
plot_perceptron(ax, data_x, data_y, w)
ax.set_title(f"w={w}, iterations={nmb_iter}")
plt.show()
```

(b):

Comment:

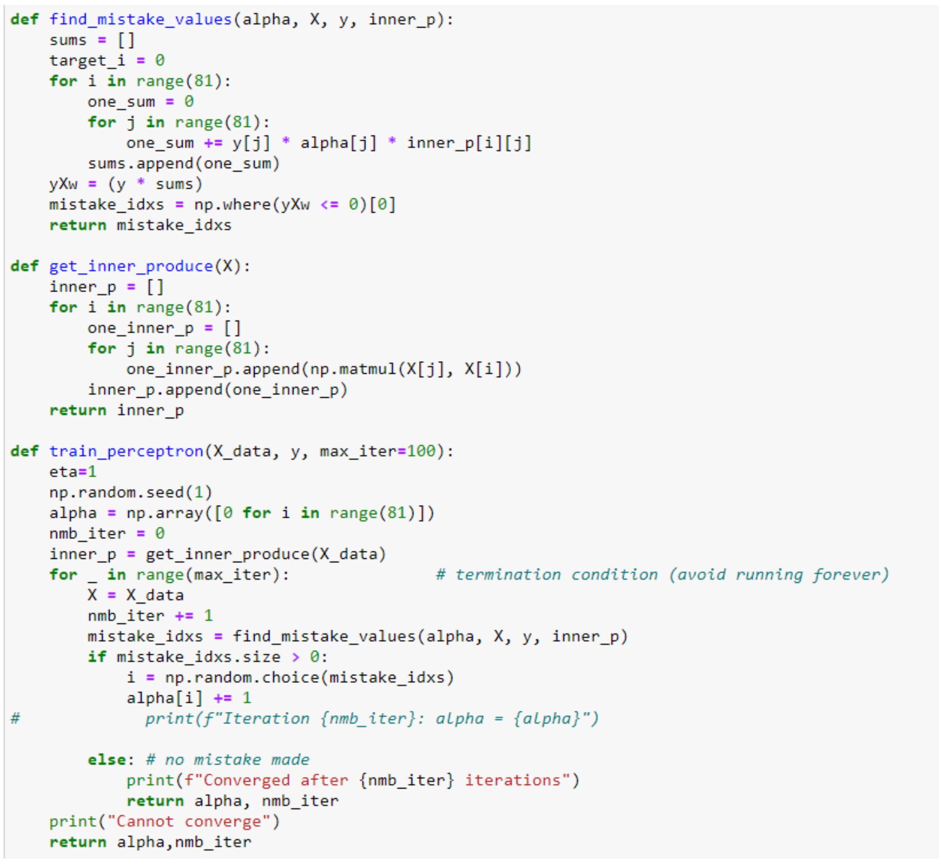
The outcomes of the two methods including number of iterations and the perceptron are the same, which makes sense. There is no essential difference between the two methods. The dual method can reduce the calculation time by calculating the inner product ahead of time.





```python
import numpy as np
import pandas as pd # not really needed, only for preference
import matplotlib.pyplot as plt
```

```python
def plot_perceptron(ax, data_X, data_y, w):
#     print(np.where(y==1)[0])
    X = pd.DataFrame(data_X)
    y = pd.DataFrame(data_y)
    pos_points = X.iloc[np.where(y==1)[0]]
    neg_points = X.iloc[np.where(y==-1)[0]]
    ax.scatter(pos_points[1], pos_points[2], color='blue')
    ax.scatter(neg_points[1], neg_points[2], color='red')
    xx = np.linspace(-6,6)
    yy = -w[0]/w[2] - w[1]/w[2] * xx
    ax.plot(xx, yy, color='orange')

    ratio = (w[2]/w[1] + w[1]/w[2])
    xpt = (-1*w[0] / w[2]) * 1/ratio
    ypt = (-1*w[0] / w[1]) * 1/ratio

    ax.arrow(xpt, ypt, w[1], w[2], head_width=0.2, color='orange')
    ax.axis('equal')
```



```python
def find_mistake_values(alpha, X, y, inner_p):
    sums = []
    target_i = 0
    for i in range(81):
        one_sum = 0
        for j in range(81):
            one_sum += y[j] * alpha[j] * inner_p[i][j]
        sums.append(one_sum)
    yXw = (y * sums)
    mistake_idxs = np.where(yXw <= 0)[0]
    return mistake_idxs

def get_inner_produce(X):
    inner_p = []
    for i in range(81):
        one_inner_p = []
        for j in range(81):
            one_inner_p.append(np.matmul(X[j], X[i]))
        inner_p.append(one_inner_p)
    return inner_p

def train_perceptron(X_data, y, max_iter=100):
    eta=1
    np.random.seed(1)
    alpha = np.array([0 for i in range(81)])
    nmb_iter = 0
    inner_p = get_inner_produce(X_data)
    for _ in range(max_iter):               # termination condition (avoid running forever)
        X = X_data
        nmb_iter += 1
        mistake_idxs = find_mistake_values(alpha, X, y, inner_p)
        if mistake_idxs.size > 0:
            i = np.random.choice(mistake_idxs)
            alpha[i] += 1
#             print(f"Iteration {nmb_iter}: alpha = {alpha}")

        else: # no mistake made
            print(f"Converged after {nmb_iter} iterations")
            return alpha, nmb_iter
    print("Cannot converge")
    return alpha,nmb_iter
```

```python
data_x = pd.read_csv('Q3X.csv')
data_y = pd.read_csv('Q3y.csv')
print(data_y.shape)
print(data_x.shape)
data_x = data_x.values
data_y = data_y.values
alpha,nmb_iter = train_perceptron(data_x, data_y, 100)
n = 81
```

```
(81, 1)
(81, 3)
Converged after 33 iterations
```

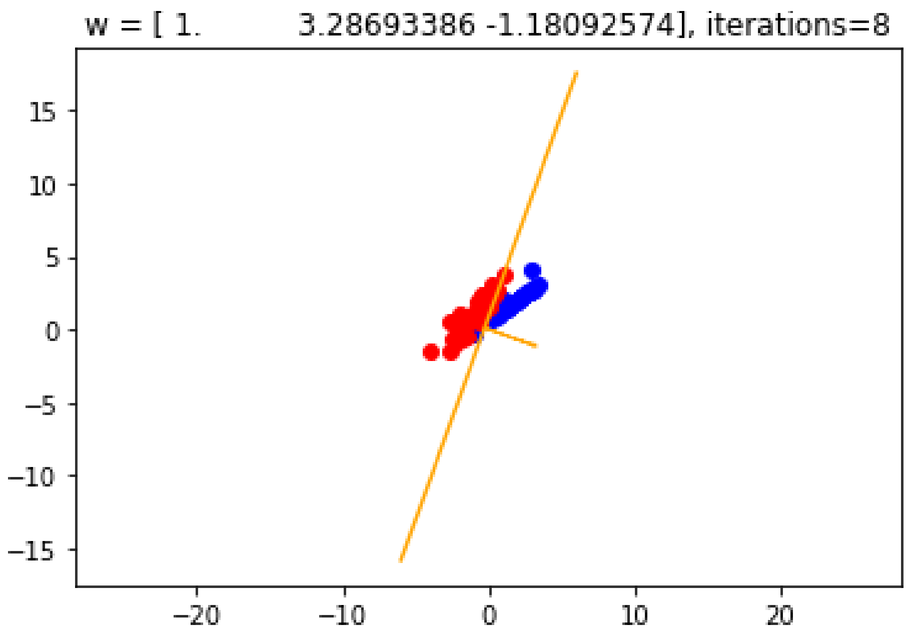
```python
print(alpha)
w = [0,0,0]
for i in range(len(alpha)):
    w[0] += (alpha[i] * data_y[i] * data_x[i])[0]
    w[1] += (alpha[i] * data_y[i] * data_x[i])[1]
    w[2] += (alpha[i] * data_y[i] * data_x[i])[2]

fig, ax = plt.subplots()
plot_perceptron(ax, data_x, data_y, w) # from neural learning lab
ax.set_title(f"alpha = {w}, iterations={nmb_iter}")
plt.savefig("name.png", dpi=300) # if you want to save your plot as a png
plt.show()

plt.plot(range(1,n+1),alpha)
plt.ylabel("Value of alpha")
plt.xlabel("Value of i")
plt.title("Dual perceptron")
plt.show()
```

```
[0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 3 1 0 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 2 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 2 0 0 0 0 0 9 0 0 0 0
 0 0 0 0 0 5 1]
```

(c):

Weight:

```
[ 1.          3.28693386 -1.18092574]
```



```python
import numpy as np
import pandas as pd # not really needed, only for preference
import matplotlib.pyplot as plt
```

```python
def plot_perceptron(ax, data_X, data_y, w):
#     print(np.where(y==1)[0])
    X = pd.DataFrame(data_X)
    y = pd.DataFrame(data_y)
    pos_points = X.iloc[np.where(y==1)[0]]
    neg_points = X.iloc[np.where(y==-1)[0]]
    ax.scatter(pos_points[1], pos_points[2], color='blue')
    ax.scatter(neg_points[1], neg_points[2], color='red')
    xx = np.linspace(-6,6)
    yy = -w[0]/w[2] - w[1]/w[2] * xx
    ax.plot(xx, yy, color='orange')

    ratio = (w[2]/w[1] + w[1]/w[2])
    xpt = (-1*w[0] / w[2]) * 1/ratio
    ypt = (-1*w[0] / w[1]) * 1/ratio

    ax.arrow(xpt, ypt, w[1], w[2], head_width=0.2, color='orange')
    ax.axis('equal')
```

```python
def find_mistake_values(w, X, y, r, III):
    sums = []
    target_i = 0
    for i in range(81):
        one_sum = y[i] * np.matmul(w, X[i]) + III[i] * r
        sums.append(one_sum)
    sums = np.array(sums)
    mistake_idxs = np.where(sums <= 0)[0]
    return mistake_idxs

def train_perceptron(X_data, y, max_iter=100, r=2):
    eta=1
    np.random.seed(1)
    III = np.array([0 for i in range(81)])
    w = np.array([0,0,0])
    nmb_iter = 0
    for _ in range(max_iter):              # termination condition (avoid running forever)
        X = X_data
        nmb_iter += 1
        mistake_idxs = find_mistake_values(w, X, y, r, III)
        if mistake_idxs.size > 0:
            i = np.random.choice(mistake_idxs)
            w = w + y[i] * X[i]
            III[i] = 1
#             print(f"Iteration {nmb_iter}: alpha = {alpha}")

        else: # no mistake made
            print(f"Converged after {nmb_iter} iterations")
            return w, nmb_iter
    print("Cannot converge")
    return w,nmb_iter
```

```python
data_x = pd.read_csv('Q3X.csv')
data_y = pd.read_csv('Q3y.csv')
print(data_y.shape)
print(data_x.shape)
data_x = data_x.values
data_y = data_y.values
w, nmb_iter = train_perceptron(data_x, data_y, 100, 2)
n = 81
```

```
(81, 1)
(81, 3)
Converged after 8 iterations
```

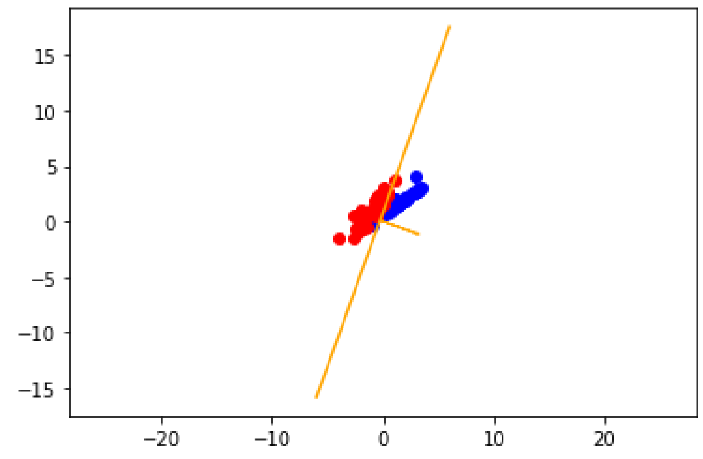
```python
fig, ax = plt.subplots()
print(w)
plot_perceptron(ax, data_x, data_y, w) # from neural learning lab
ax.set_title(f"w = {w}, iterations={nmb_iter}")
plt.savefig("name.png", dpi=300) # if you want to save your plot as a png
plt.show()
```

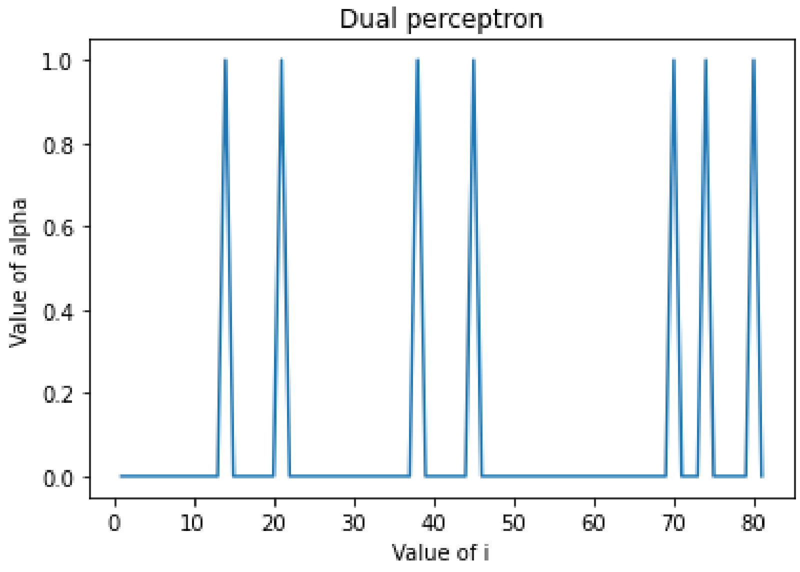
(d):

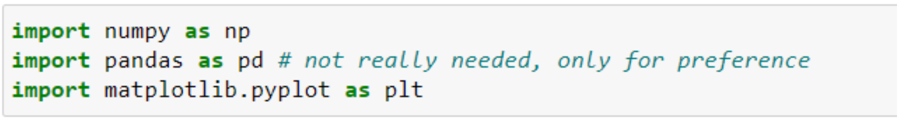
Pseudo Code:

```
Input: $(x_1, y_1), \ldots, (x_n, y_n), r > 0.$
Initialize: $\alpha^{(0)} = (0,0,\ldots,0) \in R^n, I = (0,0,\ldots,0) \in R^n$ .
For t = 1…max_iter:
    if there is an index i such that $y_i \sum_{j=1}^{n} y_j \alpha_j < x_j, x_i > + I_i r \leq 0$:
        $\alpha_i^{(t+1)} = \alpha_i^{(t)} + 1; t = t + 1; I_i = 1$
    Else:
        $output\ \ \alpha_i^{(t+1)}, t$
```

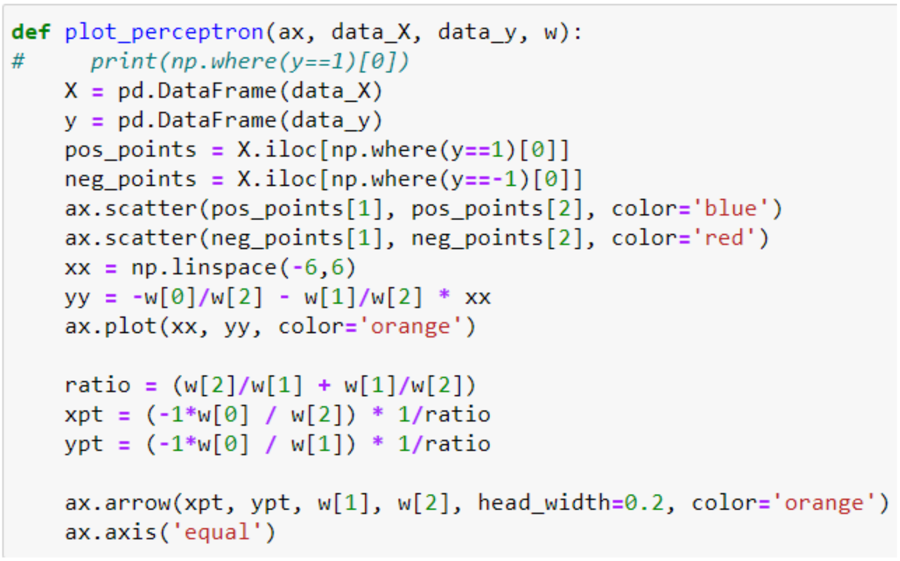
alpha = [1.0, 3.2869338591220925, -1.1809257442407457], iterations=8



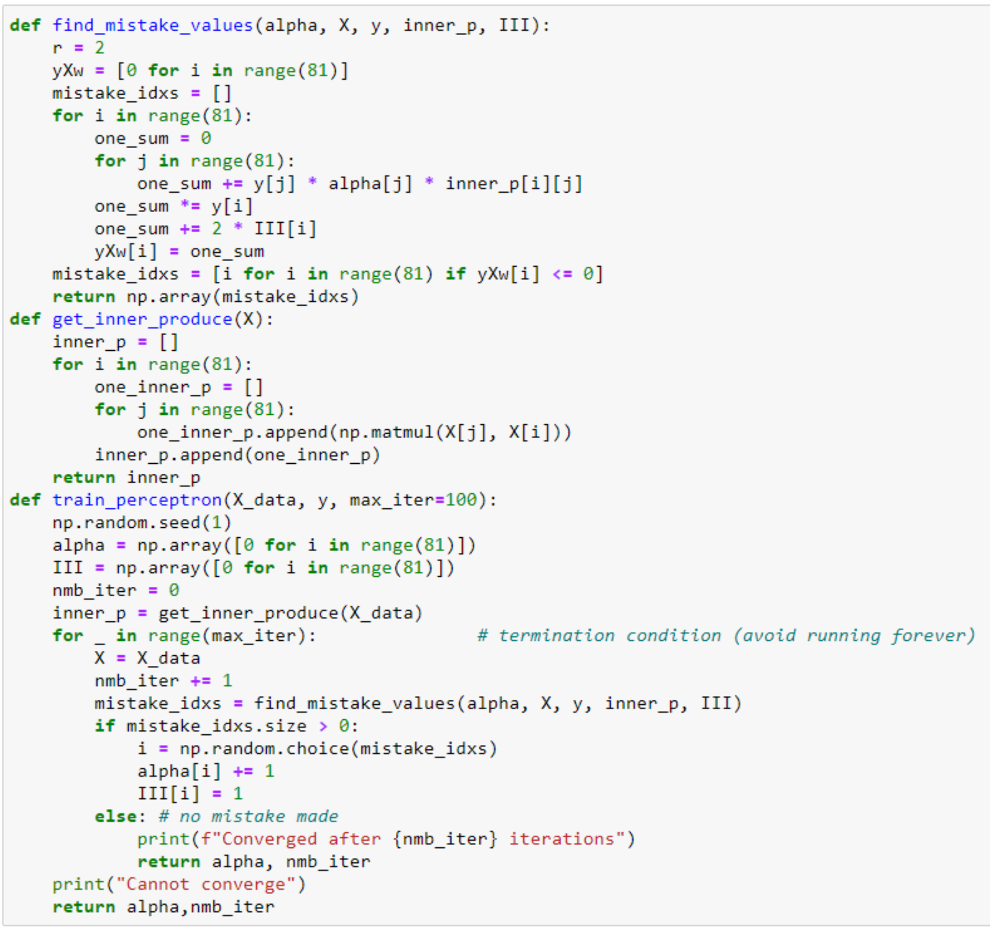




```python
import numpy as np
import pandas as pd # not really needed, only for preference
import matplotlib.pyplot as plt
```



```python
def plot_perceptron(ax, data_X, data_y, w):
#     print(np.where(y==1)[0])
    X = pd.DataFrame(data_X)
    y = pd.DataFrame(data_y)
    pos_points = X.iloc[np.where(y==1)[0]]
    neg_points = X.iloc[np.where(y==-1)[0]]
    ax.scatter(pos_points[1], pos_points[2], color='blue')
    ax.scatter(neg_points[1], neg_points[2], color='red')
    xx = np.linspace(-6,6)
    yy = -w[0]/w[2] - w[1]/w[2] * xx
    ax.plot(xx, yy, color='orange')

    ratio = (w[2]/w[1] + w[1]/w[2])
    xpt = (-1*w[0] / w[2]) * 1/ratio
    ypt = (-1*w[0] / w[1]) * 1/ratio

    ax.arrow(xpt, ypt, w[1], w[2], head_width=0.2, color='orange')
    ax.axis('equal')
```



```python
def find_mistake_values(alpha, X, y, inner_p, III):
    r = 2
    yXw = [0 for i in range(81)]
    mistake_idxs = []
    for i in range(81):
        one_sum = 0
        for j in range(81):
            one_sum += y[j] * alpha[j] * inner_p[i][j]
        one_sum *= y[i]
        one_sum += 2 * III[i]
        yXw[i] = one_sum
    mistake_idxs = [i for i in range(81) if yXw[i] <= 0]
    return np.array(mistake_idxs)
def get_inner_produce(X):
    inner_p = []
    for i in range(81):
        one_inner_p = []
        for j in range(81):
            one_inner_p.append(np.matmul(X[j], X[i]))
        inner_p.append(one_inner_p)
    return inner_p
def train_perceptron(X_data, y, max_iter=100):
    np.random.seed(1)
    alpha = np.array([0 for i in range(81)])
    III = np.array([0 for i in range(81)])
    nmb_iter = 0
    inner_p = get_inner_produce(X_data)
    for _ in range(max_iter):                   # termination condition (avoid running forever)
        X = X_data
        nmb_iter += 1
        mistake_idxs = find_mistake_values(alpha, X, y, inner_p, III)
        if mistake_idxs.size > 0:
            i = np.random.choice(mistake_idxs)
            alpha[i] += 1
            III[i] = 1
        else: # no mistake made
            print(f"Converged after {nmb_iter} iterations")
            return alpha, nmb_iter
    print("Cannot converge")
    return alpha,nmb_iter
```

```
data_x = pd.read_csv('Q3X.csv')
data_y = pd.read_csv('Q3y.csv')
print(data_y.shape)
print(data_x.shape)
data_x = data_x.values
data_y = data_y.values
alpha,nmb_iter = train_perceptron(data_x, data_y, 100)
n = 81
```

```
(81, 1)
(81, 3)
Converged after 8 iterations
```

```
print(alpha)
w = [0,0,0]
for i in range(len(alpha)):
    w[0] += (alpha[i] * data_y[i] * data_x[i])[0]
    w[1] += (alpha[i] * data_y[i] * data_x[i])[1]
    w[2] += (alpha[i] * data_y[i] * data_x[i])[2]

fig, ax = plt.subplots()
plot_perceptron(ax, data_x, data_y, w) # from neural learning lab
ax.set_title(f"alpha = {w}, iterations={nmb_iter}")
plt.savefig("name.png", dpi=300) # if you want to save your plot as a png
plt.show()

plt.plot(range(1,n+1),alpha)
plt.ylabel("Value of alpha")
plt.xlabel("Value of i")
plt.title("Dual perceptron")
plt.show()
```

(e):
The effect: Once one index has been updated, it will become hard for it to get updated again. It must have a value smaller than -2 since I plus 1*2 to it in next iteration. Through this, I can avoid updating the same index in different iterations.
When the dataset contains some noise data, this algorithm performs better since it can prevent one data been updated in the perceptron multiple times. By printing out the alpha list from (b) and (d).
(b): [0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 3 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 2 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 2 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 5
1]
(d): [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1
0]
We can see that in (b) there are data points being updated 9 and 5 times. But instead in (b), they are only been updated once.

The disadvantage is when the values in the dataset is small enough which means we indeed need to update one data multiple times but we do not actually do that. The algorithm may stop us from doing that. It may stop us from getting a convergent solution or we have not gotten a good model but the algorithm stops.