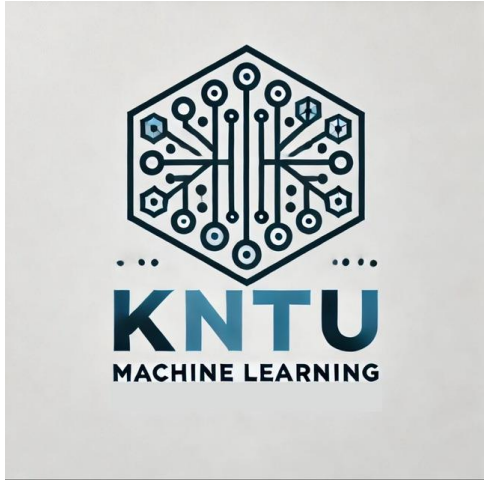
K. N. Toosi
University of Technology

HomeWork3

Prerequisites for Machine Learning

Professor: **Dr.B.Nasersharif**

Provider: **Mehran Tamjidi**



Telegram Group: t.me/ML_403_1

Question 1

Question 1 (30 marks)

Use the perceptron learning algorithm with $\eta = 1$ to compute a plane that linearly separates A+ and A- defined in the below:

A+ = {(1,1),(0,2),(3,0)} A-= {(-2,-1),(0,-2)}

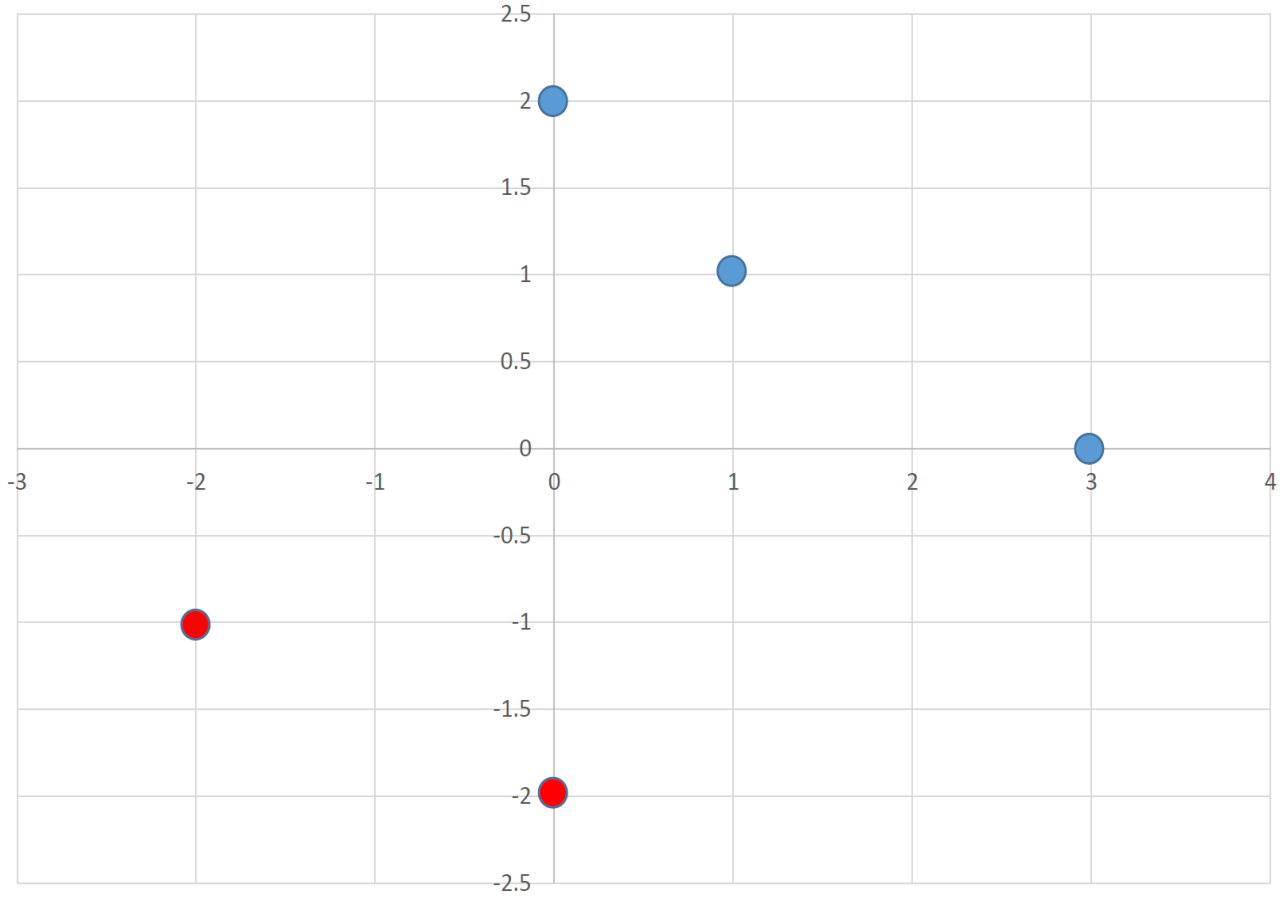
(a) Let w(0) be the zero vector. Modify the algorithm so that b is computed together with the components of w. This can be done by adding an extra component of "1" to each training vector, and then computing w' = (w1, w2, b). Draw the obtained separating line along with the dataset.

(b) Solve this problem using the Least Square method. Draw your obtained line in the drawing of part (a) and compare your lines.

(c) Solve this problem using the Fisher method and draw the separating line as explained in (b).

Lets see the separability of our dataset

Each color corresponds to a specific type of target data



Simple perceptron model can separate this dataset. Because it is linearly separatable.

part A

Perceptron Learning Algorithm

$$E_p(w) = -\sum_{n \in M} w^t x_n t_n$$

condition: $W^t x_n t_n \geq 0 \; \forall n$

$$\frac{dE_p(w)}{dw} = -\sum_{n \in M} x_n t_n$$

$$w^{t+1} = w^t - \eta \nabla E_p(w) = w^t + \eta \sum_{n \in M} x_n t_n$$

X			T
x_0	x_1	x_2	
1	1	1	1
1	0	2	1
1	3	0	1
1	-2	-1	-1
1	0	-2	-1

$Y =$

x_0	x_1	x_2
1	1	1
1	0	2
1	3	0
-1	2	1
-1	0	2

labels

tn xn

$W_0 = (w_1, w_2, b) = (0, 0, 0)$

first iteration: $W^0 x_{sTJ} = -.5 < 0 \Rightarrow$ we need to update weights.

$Y_{(1)} = [0\ 0\ 0][1\ 1\ 1]^T \rightarrow$ updating $W_1 = [0\ 0\ 0] + 2[1\ 1\ 1] = [1\ 1\ 1]$

$Y_{(2)} = [1\ 1\ 1][1\ 0\ 2]^T > 0 \rightarrow$ no update

$Y_{(3)} = [1\ 1\ 1][1\ 3\ 0]^T > 0 \rightarrow$ no update

$Y_{(4)} = [1\ 1\ 1][-1\ 2\ 1]^T > 0 \rightarrow$ no update

$Y_{(5)} = [1\ 1\ 1][1\ 0\ 2]^T > 0 \rightarrow$ no update

So the final weight of matrix is $[1\ 1\ 1]$ because of that it doesn't change each epoch.

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\downarrow$$

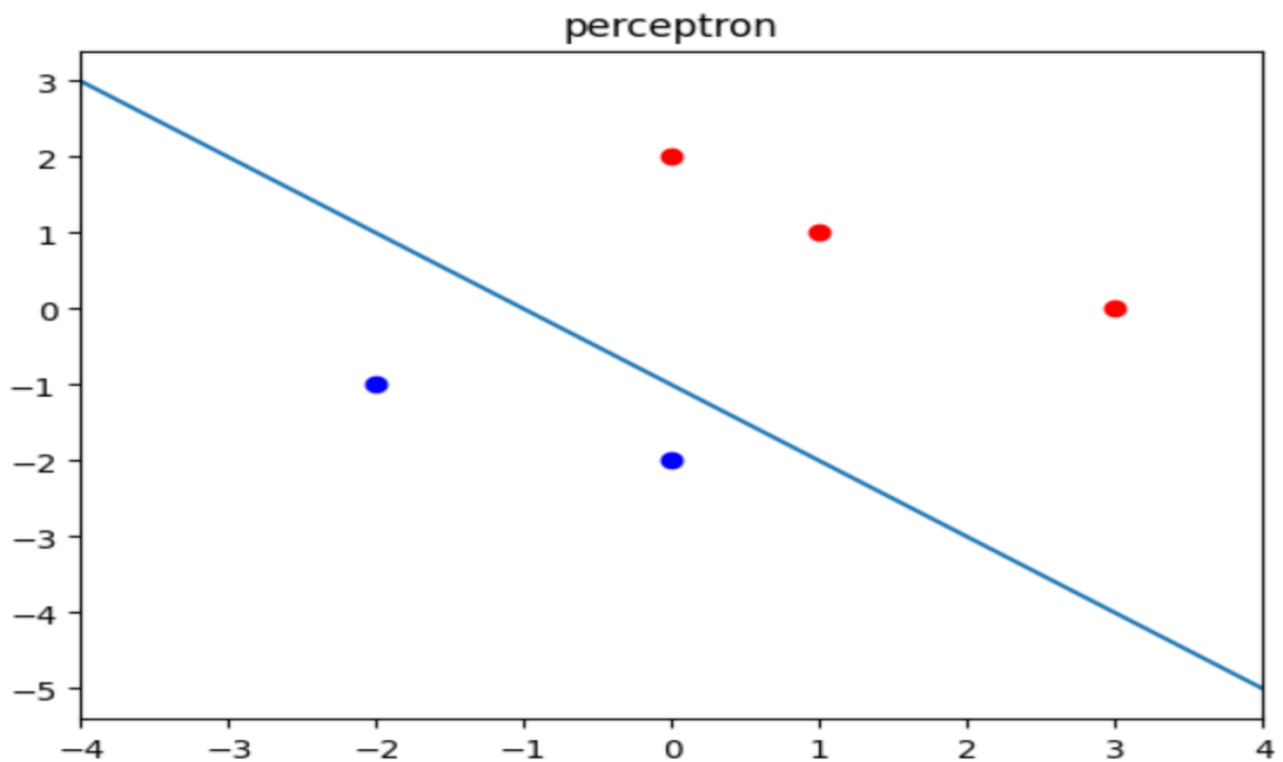
$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

$$\sum_{i=1}^{N} w_i x_i + b = 0 \rightarrow b = w_0 \rightarrow x_1 + x_2 + 1 = 0 \rightarrow \boxed{x_2 = -x_1 - 1}$$

Seperation Line

In this case we see the plot separation line obtained by perceptron.

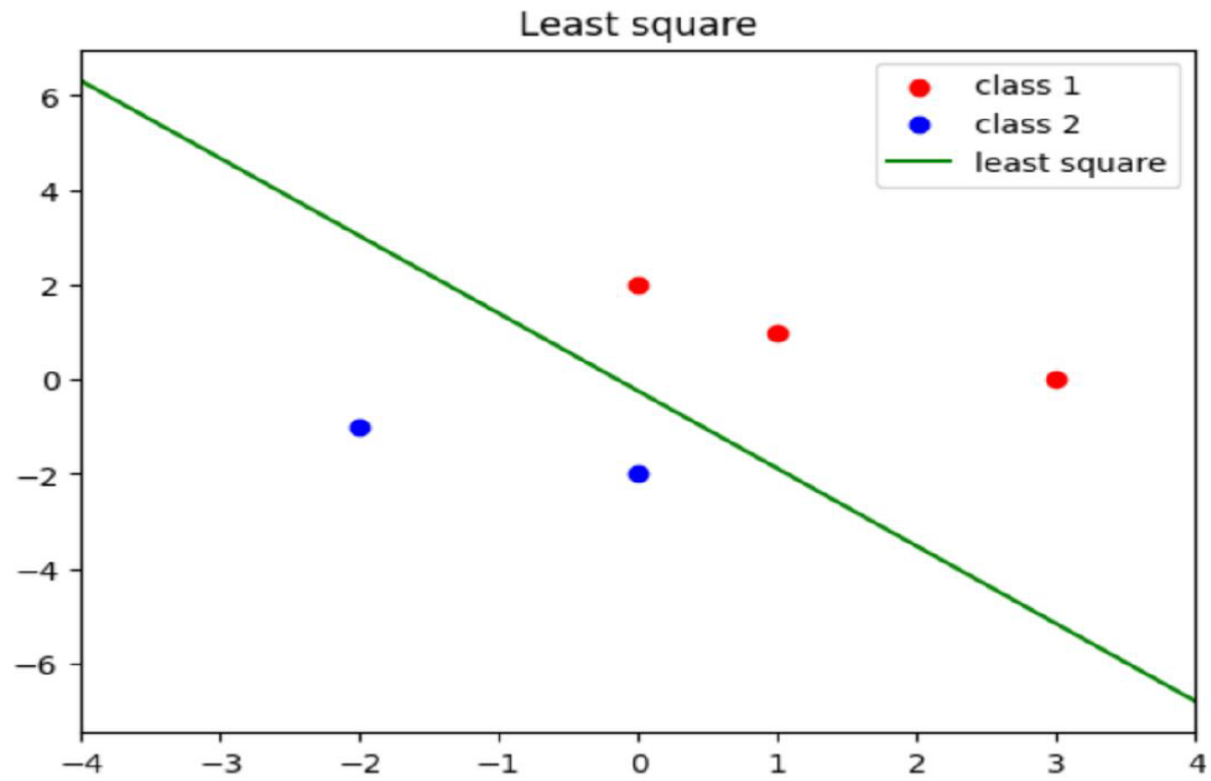
```
d=np.array([[1,1, 1],[1, 0, 2],[1, 3, 0],[-1, -2, -1], [-1, 0, -2]])
d.shape
plt.scatter(d[:3,1],d[:3,2],c='red')
plt.scatter(d[3:,1],d[3:,2],c='blue')
x1=np.linspace(-4, 4 ,100)
x2=-1*x1+-1
plt.plot(x1,x2)
plt.xlim(-4,4)
plt.title('perceptron')
plt.show()
```



part B

In this section we calculate the weight matrix with least square method.

```
d=np.array([[1,1, 1],[1, 0, 2],[1, 3, 0],[-1, -2, -1], [-1, 0, -2]])
d.shape
plt.scatter(d[:3,1],d[:3,2],c='red',label='class 1')
plt.scatter(d[3:,1],d[3:,2],c='blue',label='class 2')
x1=np.linspace(-4, 4 ,100)
x2=-1.64*x1+-0.2473
plt.plot(x1,x2,c='green',label='least square')
plt.xlim(-4,4)
plt.title('Least square')
plt.legend()
plt.show()
```



Least squared method

$$\tilde{W} = \left(\tilde{X}^t \tilde{X}\right)^{-1} \tilde{X} T$$

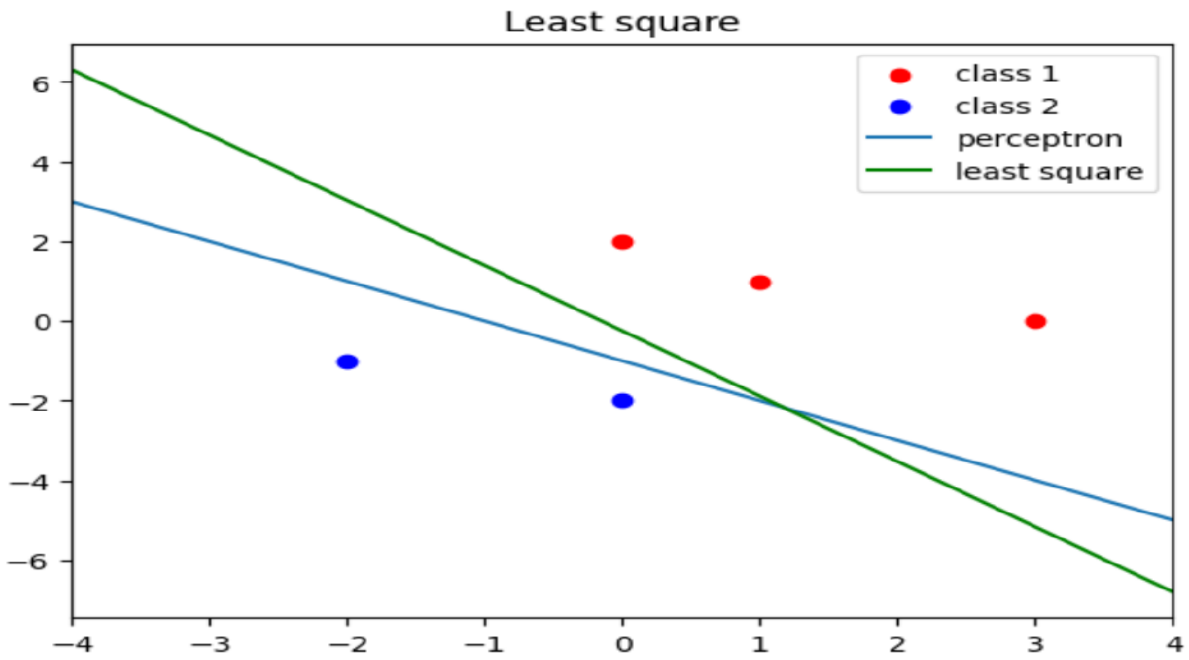
part B

Guess which one is better?

Upon the initial weight choice, its answers vary with each set of initial weights. However, generally, the perceptron is better than the least squares method in some cases because, when the data is linearly separable, it guarantees finding a separating line, whereas the least squares method fails in such cases. Conversely, in other cases, such as here, the least squares method is a better option for providing a better margin.

Now lets compare least square and perceptron.

```
d=np.array([[1,1, 1],[1, 0, 2],[1, 3, 0],[-1, -2, -1], [-1, 0, -2]])
d.shape
plt.scatter(d[:3,1],d[:3,2],c='red',label='class 1')
plt.scatter(d[3:,1],d[3:,2],c='blue',label='class 2')
x1=np.linspace(-4, 4 ,100)
x2=-1*x1+-1
plt.plot(x1,x2,label='perceptron')
x2=-1.64*x1+-0.2473
plt.plot(x1,x2,c='green',label='least square')
plt.xlim(-4,4)
plt.title('Least square')
plt.legend()
plt.show()
```



Its seems that in this particular case least square has better solution because it provided better margin. Because perceptron models answer is severely dependent

part C

Fisher method computations

We First compute mean for each class.

$\mu_1 = \text{mean}(C_1) = [1,3\overline{3}\ ,\ 1]$

$\mu_2 = \text{mean}(C_2) = [-1\ ,\ -1,5]$

Then we compute Covariance matrix for each class:

$S_1 = 3 \times \text{Cov}(C_1) = \begin{bmatrix} 2,3\overline{3} & -1,5 \\ -1,5 & 1 \end{bmatrix}$

$S_2 = 2 \times \text{Cov}(C_2) = \begin{bmatrix} 2 & -1 \\ -1 & 1,5 \end{bmatrix}$

$S_W = S_1 + S_2 = \begin{bmatrix} 4,3\overline{3} & -2,5 \\ -2,5 & 1,5 \end{bmatrix}$

$S_W = S_W^{-1} = \begin{bmatrix} 6 & 10 \\ 10 & 17,3\overline{3} \end{bmatrix}$

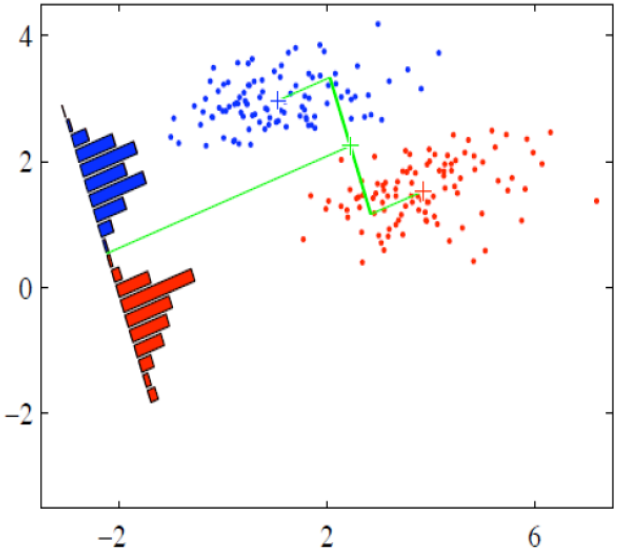
$W = S_W^{-1}(\mu_1 - \mu_2) \Rightarrow \begin{bmatrix} 6 & 10 \\ 10 & 17,3\overline{3} \end{bmatrix} \begin{bmatrix} 2,3\overline{3} \\ 2,5 \end{bmatrix} \Rightarrow \begin{bmatrix} 39 \\ 66\overline{16} \end{bmatrix}$

Normalizing $\Rightarrow \frac{W}{\|W\|} = \begin{bmatrix} 0,5049 \\ 0,8631 \end{bmatrix}$

$\sum_{i=0}^{1} w_i x_i = 0 \longrightarrow 0,5049 x_1 + 0,8631 x_2 = 0$

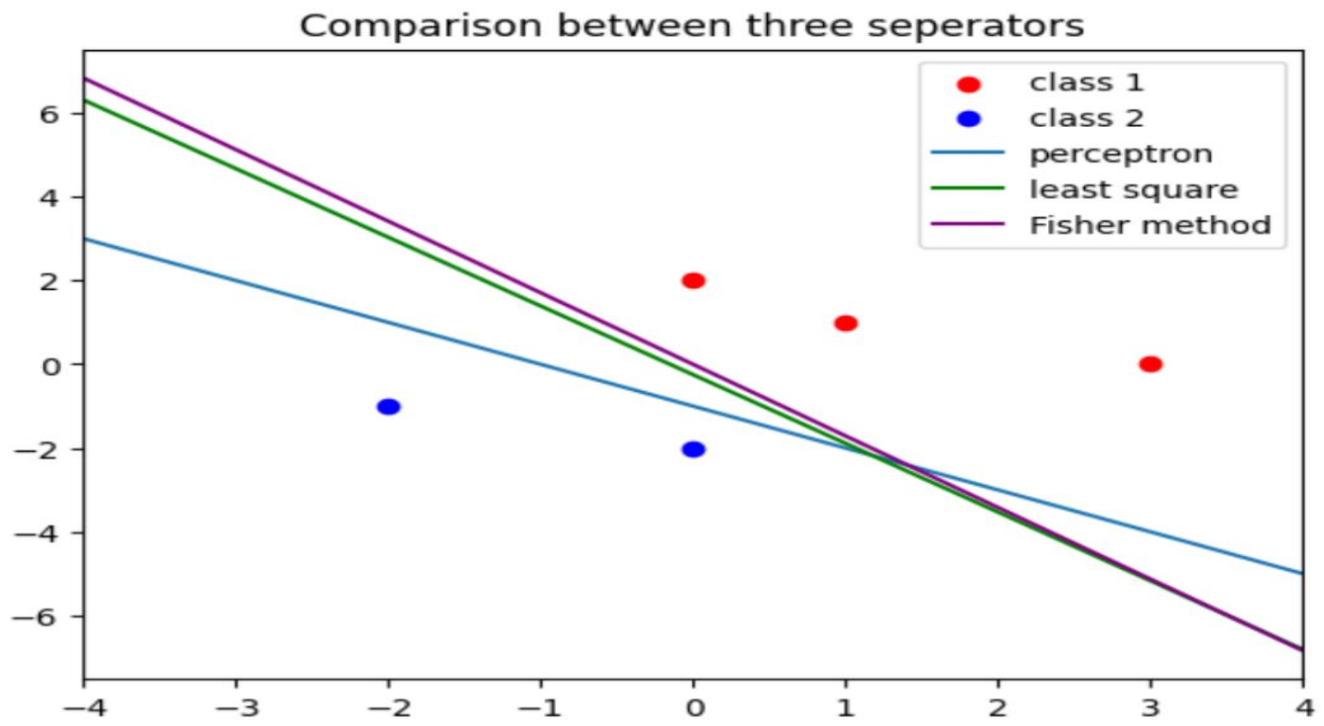
Finally $\boxed{x_1 = -1,709 x_2}$

and then we can plot this line with matPlotLib.



And finally we provide a fair comparasion between this three mehtods for this two class dataset.

```
d=np.array([[1,1, 1],[1, 0, 2],[1, 3, 0],[-1, -2, -1], [-1, 0, -2]])
d.shape
plt.scatter(d[:3,1],d[:3,2],c='red',label='class 1')
plt.scatter(d[3:,1],d[3:,2],c='blue',label='class 2')
x1=np.linspace(-4, 4 ,100)
x2=-1*x1+-1
plt.plot(x1,x2,label='perceptron')
x2=-1.64*x1+-0.2473
plt.plot(x1,x2,c='green',label='least square')
x2=-1.709*x1
plt.plot(x1,x2,c='purple',label='Fisher method')
plt.xlim(-4,4)
plt.title('Comparison between three seperators')
plt.legend()
plt.show()
```



Question 2

Question 2 (20 marks)

Given a set of data points $\{\mathbf{x}_n\}$, we can define the *convex hull* to be the set of all points $\mathbf{x}$ given by

$$x = \sum_n \alpha_n x_n$$

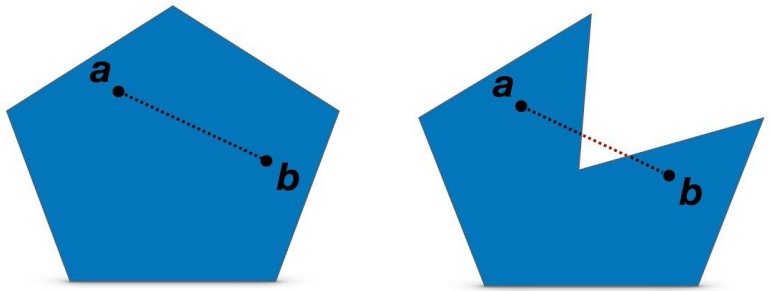
Where $\alpha_n \geq 0$ *and* $\sum_n \alpha_n = 1$.

Consider a second set of points $\{\mathbf{y}_n\}$ together with their corresponding convex hull. By definition, the two sets of points will be linearly separable if there exists a vector $\mathbf{w}$ and a scalar w_0 such that $w^T x_n + w_0 > 0$_ for all $\mathbf{x}_n$, and _$w^T y_n + w_0 < 0$ r for all $\mathbf{y}_n$.

Show that if their convex hulls intersect, the two sets of points cannot be linearly separable, and conversely if they are linearly separable, their convex hulls do not intersect (20 Mark).

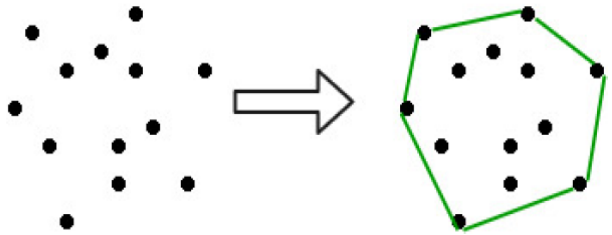
Lets learn more about convexity

What is a Convex Set?



Convex Not Convex

At the first what is convex hull?!



Convex hull is the smallest convex set which includes all of the cardinality of a set.

We want to prove that if two convex hull cardinalities have intersection, we cannot separate them with a line (they are not linearly separatable) and Vis versa.

Affine combination of points *A* and *B*

$$\lambda A + \mu B, \qquad \lambda + \mu = 1$$

Convex combination of points A and B

$$\lambda A + \mu B, \qquad \lambda + \mu = 1, \qquad \lambda,\ \mu \geq 0$$

Generalization: Euclidean n-dim space E^n

Affine combination:

$$\lambda_1 A_1 + \lambda_2 A_2 + \ldots + \lambda_k A_k, \quad \lambda_1 + \lambda_2 + \ldots + \lambda_k = 1$$

Convex combination:

$$\lambda_1 A_1 + \lambda_2 A_2 + \ldots + \lambda_k A_k, \quad \lambda_1 + \lambda_2 + \ldots + \lambda_k = 1,$$

$$\lambda_1, \ldots, \lambda_k \geq 0$$

Convex set is all of the convex combinations

Lets learn more about convexity

Convex combination

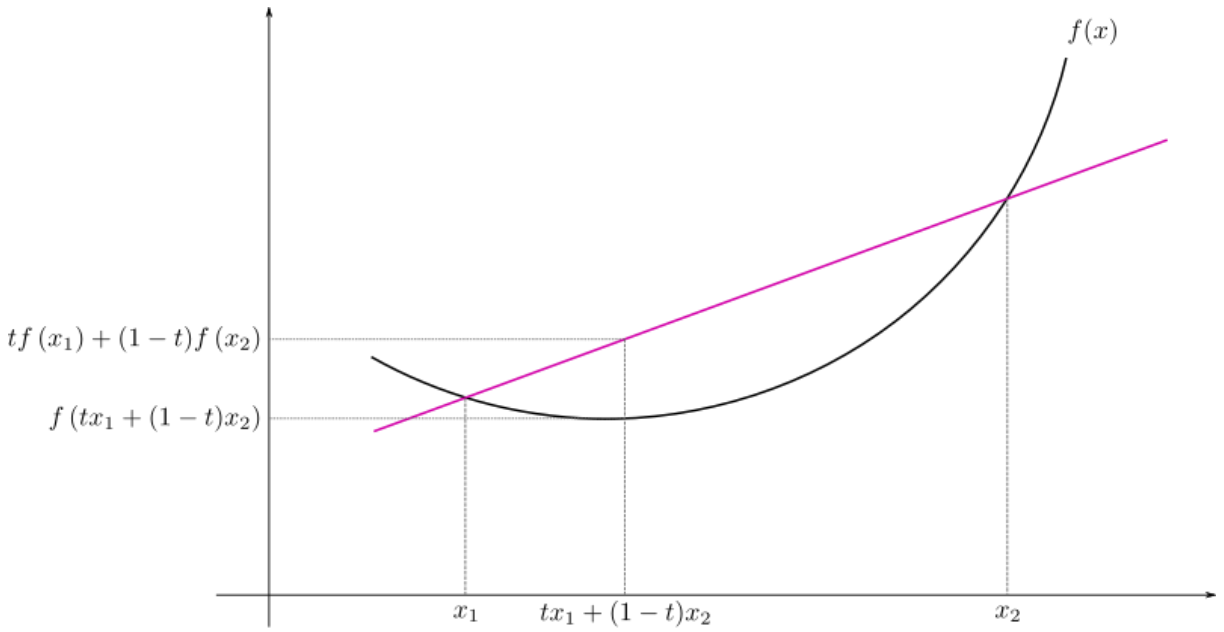
A convex combination of points $x_1, x_2, \ldots, x_n$ in a vector space is defined as:

$$x = \sum_{i=1}^{n} \lambda_i x_i$$

where:

- $\lambda_i \geq 0$ (non-negative coefficients),
- $\sum_{i=1}^{n} \lambda_i = 1$ (weights sum to 1).

Convex function



Suppose that we have two convex set

$$x = \sum_n a_n x_n \longrightarrow \begin{cases} S_{x_n} = \sum a_n (w^T x_n + w_0) \\ S_{x_p} = \sum \beta_n (w^T y_n + w_0) \end{cases}$$

$\exists t$ which can be in both set S_{x_n} and S_{x_p}

$$\circledast \quad S_t = \sum a_n (w^T t + w_0) = \sum \beta_n (w^T t + w_0)$$

to be linearly seperatable we have to find w and w_0.

because $\beta_n, \alpha_n \geq 0$ we cannot find any seperation Line which satisfies the following condition for all x_n & y_n (including t)

$$\begin{cases} w^T x_n + w_0 > 0 \\ w^T y_n + w_0 < 0 \end{cases}$$

and finally we cannot find a seperatio Line to seperate two sets which intersects

Part 1 Proved

to Prove the reverse of above we suppose that there is a separation line

So the we have following

$$\begin{cases} w^T a_n + w_0 > 0 \\ w^T y_n + w_0 < 0 \end{cases}$$

but in this case we cannot find common cardinality like t that satisfies equation ✳ and we reach contradiction in this case.

Part 2 is proved too..

Question 3 (30 marks)

In many real-world scenarios, our data has millions of dimensions, but a given example has only hundreds of non-zero features. For example, in document analysis with word counts for features, our dictionary may have millions of words, but a given document has only hundreds of unique words. In this question, we will make l2 regularized (stochastic gradient descent (SGD) efficient when our input data is sparse. Recall that in l2 regularized logistic regression, we want to maximize the following objective (in this problem we have excluded w0 for simplicity):

$$F(\mathbf{w}) = \frac{1}{N}\sum_{j=1}^{N} l(\mathbf{x}^{(j)}, y^{(j)}, \mathbf{w}) - \frac{\lambda}{2}\sum_{i=1}^{d} \mathbf{w}_i^2$$

$$l(\mathbf{x}^{(j)}, y^{(j)}, \mathbf{w}) = y^{(j)}(\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i^{(j)}) - \ln(1 + \exp(\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i^{(j)}))$$

Where $l(x^{(i)}, y^{(j)}, w)$ is the logistic objective function and the remaining sum is our regularization penalty.

When we do stochastic gradient descent on point $(x^{(i)}, y^{(j)})$ are approximating the objective function as

$$F(\mathbf{w}) \approx l(\mathbf{x}^{(j)}, y^{(j)}, \mathbf{w}) - \frac{\lambda}{2}\sum_{i=1}^{d} \mathbf{w}_i^2$$

part A

$$F(w) = \frac{1}{N}\sum_{j=1} l(x^{(i)}, y^{(i)}, w) - \frac{\lambda}{2}\sum_{j=1}^{d} w_i^2$$

$$F(w) = \frac{1}{N}\sum_{j}\left(y^{(i)}\left(\sum_{j=1}^{d} w_i x_i^{(j)}\right) - \ln\left(1 + \exp\left(\sum_{i=1}^{d}(w_i x_i^{(j)})\right)\right) - \frac{\lambda}{2}\sum_{j=1}^{d} w_i^2\right)$$

$$\text{PART A} \longrightarrow \frac{\partial F(w)}{\partial w_i} \xrightarrow{\lambda = 0} \frac{\partial F(w)}{\partial w}\left(\frac{1}{N}\sum_{j}\left(y^{(i)}\left(\sum_{i=1}^{d} w_i x_i^{(j)}\right) - \ln\left(1 + e^{\sum_{i=1}^{d}(w_i x_i^{(j)})}\right)\right)\right)$$

$$\lambda = 0$$

$$\nabla_i = \frac{\partial F(w)}{\partial w_i} = \frac{1}{N}\sum_{j}\left(y^{(j)} x_i^{(j)} - \frac{e^{\sum_{i=1}^{d}(w_i x_i^{(j)})}}{1 + e^{\sum_{i=1}^{d}(w_i x_i^{(j)})}} x_i^{(j)}\right) \quad \text{if } x_i^{(j)} \neq 0$$

$$\Longrightarrow W_{i_{k+1}} = W_{i_k} - \zeta\left(y^{(j)} x_i^{(j)} - \frac{e^{\sum_{i=1}^{d} w_i x_i^{(j)}}}{1 + e^{\sum_{i}^{d}(w_i x_i^{(j)})}} x_i^{(j)}\right)$$

which means that we only update the weights when $x_i^{(j)} \neq 0$.

↳ lower computation

M.Tamjidi

part B

$$\lambda \neq 0 \;\&\; \lambda > 0 \Rightarrow \nabla g = \frac{\partial F(w)}{\partial w_i} \xrightarrow{\lambda > 0} \Rightarrow$$

$$\nabla g_2 \Rightarrow \frac{1}{N} \sum_j \left(y^{(i)(j)} x_i^{(j)} - \frac{\exp\left(\sum_{j=1}^{d} w_i x_i^{(j)}\right) x_i^{(j)}}{1 + \exp\left(\sum_{j=1}^{d} (w_i x_i^{(j)})\right)} - \lambda \sum_{i=1}^{d} w_i \right)$$

for the sparse condition $\Rightarrow$

$$\boxed{w_i^{(k+1)} = w_i^{(k)} - \eta \nabla g_2}$$

$$w_i^{(k+1)} = w_i^{(k)} - \eta \left(y^{(j)(j)} x_i - \frac{\left(e^{\sum_{i=1}^{d} w_i x_i^{(j)}}\right) x_i^{(j)}}{1 + e^{\sum_{i=1}^{d} w_i x_i^{(j)}}} - \lambda \sum_{i=1}^{d} w_i \right)$$

in this case similar to Part A we only update weights when $x_i^{(j)} = 0$.

part C

★ ANSWER: The time complexity to calculate $\sum_k \mathbf{w}_k x_k^{(j)}$ is $O(d)$ when the data structure is dense, and $O(s)$ when the data structure is sparse. Note that even if we update $\mathbf{w}_i$ for all i, we only need to calculate $\sum_k \mathbf{w}_k x_k^{(j)}$ once, and then update the $\mathbf{w}_i$ such that $\mathbf{x}_i^{(j)} \neq 0$. So the answer is $\boxed{O(d)}$ for the dense case, and $\boxed{O(s)}$ for the sparse case.

Question 4

Question 4 (60 marks) (implementation)

You were given a dataset "dataset_i.csv" (extracted from the IRIS dataset) including 2 features and 3 classes.

(a) Consider only the first two classes (class 0, and class 1). Perform Fisher, perceptron, and least square classification methods for these two classes. Don't use ML libraries for classifiers and implement them yourself. Draw the obtained separating lines and dataset and compare among 3 resulting lines.
Also, calculate the accuracy of classifiers and compare them in a table. (15 Mark)

(b) Consider two latter classes (class 1, and class 2), Perform Fisher, perceptron, and least square classification methods for these two classes. Don't use ML libraries for classifiers and implement them yourself. Draw the obtained separating lines and dataset and compare among 3 resulting lines.
Also, calculate the accuracy of classifiers and compare them in a table. (15 Mark)

(c) Comparing (a) and (b), analyze the results and conclude in two or three sentences (10 Mark)

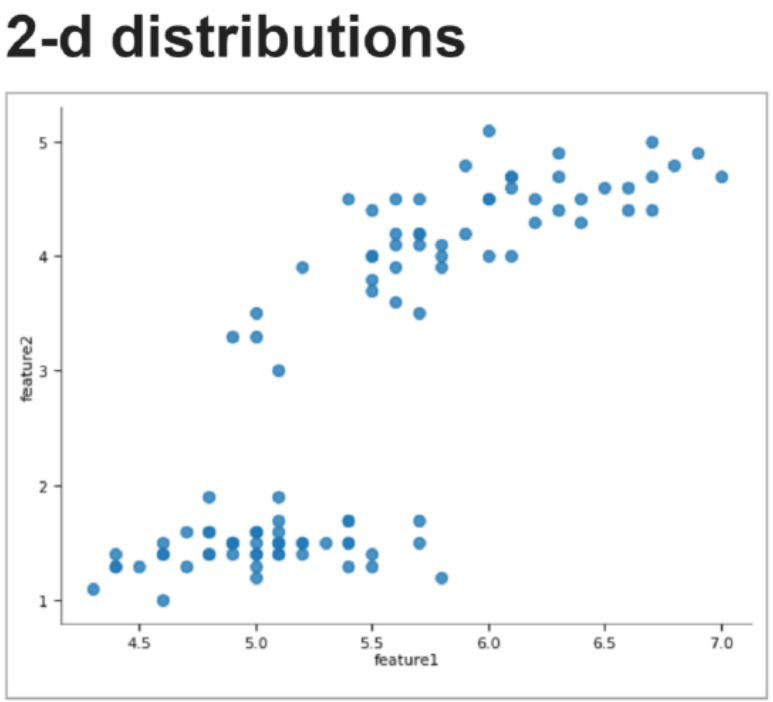
(d) Do the parts (a) and (b) for the logistic classifier. In this case, you can use ML libraries that exist for logistic classifiers. Compare the accuracy of this classifier with parts (a) and (b). (10 Mark)

(e) Perform classification for all three classes using the least square method and calculate the accuracy (10 Mark)

part A_1 (perceptron)

in the first part we only use the first two classes and we changed their labels to positive and negative pair (to easily using the sign for modifying the line!)

As we see the data linearly separatable, therefore we can find a solution with perceptron which separate all of the data.



for that aim we write down a class for calcuating the weights with perceptron:

the variable Y is the prepared dataset matrix for updating (like the exmaple in the book).

```
#inspired by theb Numerical example
class linear_perceptron:
  weights=dict()
  def __init__(self, Y):
    self.Y =Y # we append bais as the first feature of dataset!
    self.dim = self.Y.shape
  def initial_weights(self):
    W = np.random.randn(self.dim[1])
    self.weights['W0']=W
  def updating_weights(self, etha,epoches):
    self.initial_weights()
    j=0
    for epoch in range(0,epoches):
      for i in range(0, self.dim[0]):
        WY = np.dot(self.weights['W'+str(j)], self.Y[i,:])
        if WY<=0: #when we have miss labels
          self.weights['W' +str(j+1)] = self.weights['W'+ str(j)]+ etha * Y[i,:]
          j+=1
    print('weights=',self.weights['W' +str(j)],'    numbers of update=',j)
    return self.weights['W' +str(j)]

l=linear_perceptron(Y)
final_weights=l.updating_weights(1,100)
l.weights
l.dim
w=[-final_weights[0]/final_weights[2] ,-final_weights[1]/final_weights[2]]

plt.scatter(X[0:50,0],X[0:50,1] ,c='red',label='class 1')
plt.scatter(X[50:,0],X[50:,1] ,c='blue',label='class 2')
x1=np.linspace(-20, 20 ,200)
x2=w[1]*x1+w[0]
plt.plot(x1,x2,c='green',label='perceptron')
plt.title('perceptron')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()
plt.show()
```

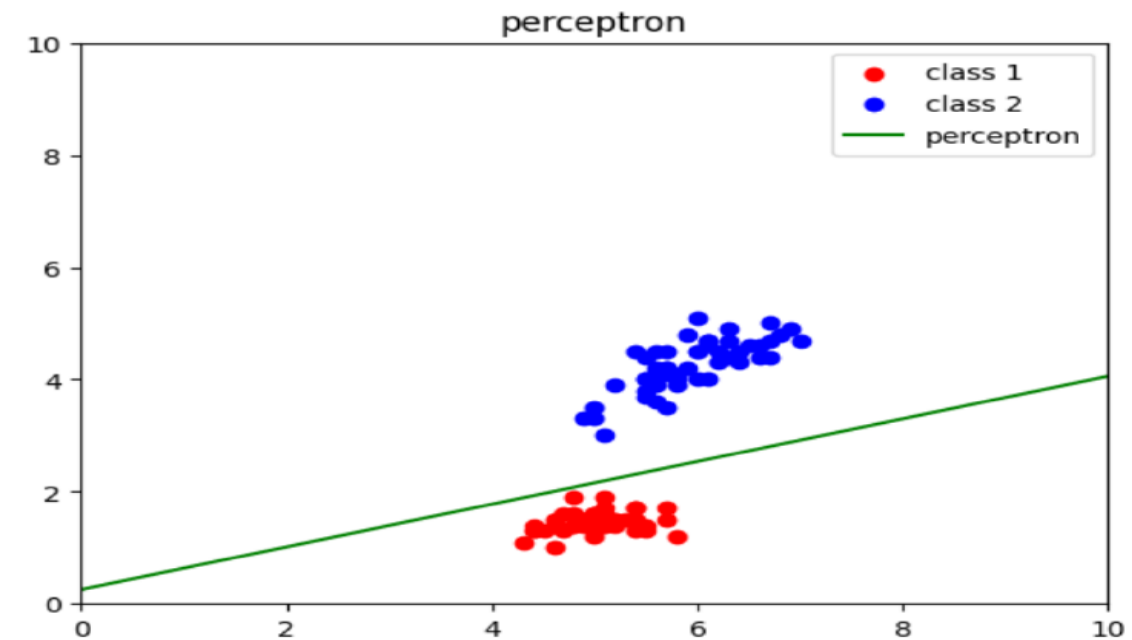
And here we can see it's convergance with only a few update (just 10).

Due to the random initialization it's answer is no longer unit and as noted in question 1, heavily depends upon the initialation weights!

You can also see it's weight in the subscript!

```
final_weights=l.updating_weights(1,100)
l.weights
l.dim
w=[-final_weights[0]/final_weights[2] ,-final_weights[1]/final_weights[2]]

plt.scatter(X[0:50,0],X[0:50,1] ,c='red',label='class 1')
plt.scatter(X[50:,0],X[50:,1] ,c='blue',label='class 2')
x1=np.linspace(-20, 20 ,200)
x2=w[1]*x1+w[0]
plt.plot(x1,x2,c='green',label='perceptron')
plt.title('perceptron')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()
plt.show()
```

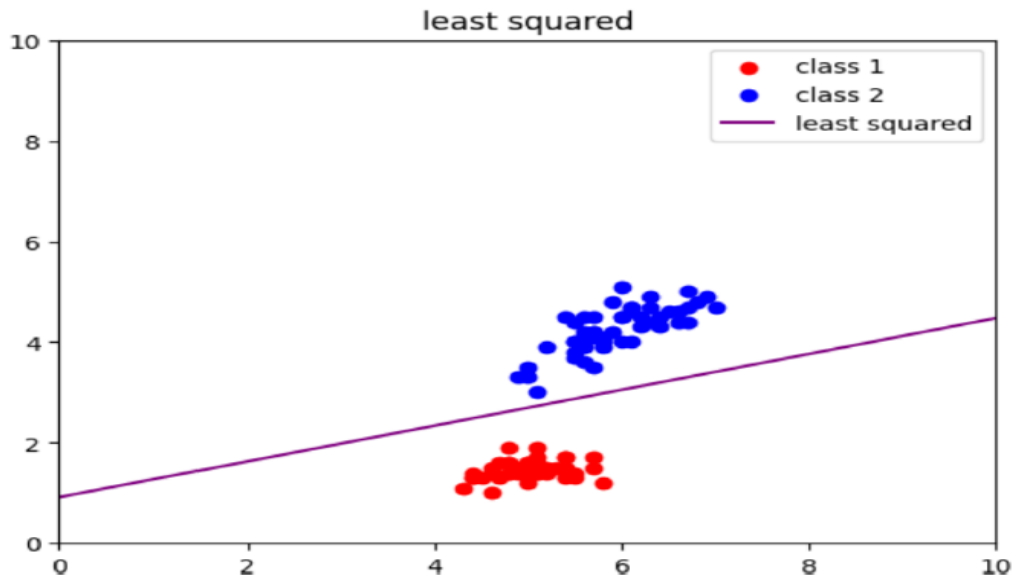
```
weights= [ 2.29612028  3.46105527 -9.09488654]     numbers of update= 10
```



part A_2(least squared method)

```
b=np.ones((label.shape))
W=np.linalg.inv((Y.T@Y))@Y.T@b
print('least squared weights=',W)
w=[-W[0]/W[2] ,-W[1]/W[2]]
plt.scatter(X[0:50,0],X[0:50,1] ,c='red',label='class 1')
plt.scatter(X[50:,0],X[50:,1] ,c='blue',label='class 2')
x1=np.linspace(-20, 20 ,200)
x2=w[1]*x1+w[0]
plt.plot(x1,x2,c='purple',label='least squared')
plt.title('least squared')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()
plt.show()
```

```
least squared weights= [[ 0.70498922]
 [ 0.27494856]
 [-0.7719192 ]]
```



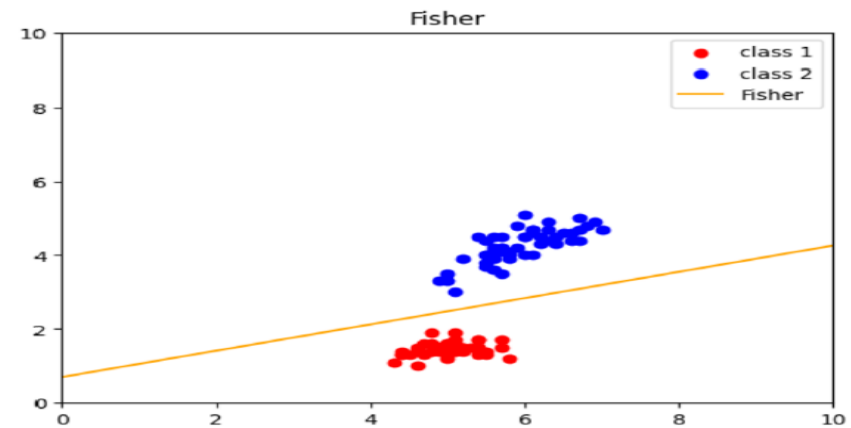
part A_3(Fisher)

here we implement Fisher algorithm for finding separator line.

```
A_plus = X[0:50,:]
A_minus = X[50:,:]

m1 = np.mean(A_plus, axis=0)
m2 = np.mean(A_minus, axis=0)

S1 = np.cov(A_plus, rowvar=False)
S2 = np.cov(A_minus, rowvar=False)

Wf = np.linalg.inv(S1 + S2).dot(m1 - m2)

Wf /= np.linalg.norm(Wf)

# Print the separating line
print(f" fisher separating line: {Wf[0]} * x + {Wf[1]} * y = 0")
print(S1)

plt.scatter(X[0:50,0],X[0:50,1] ,c='red',label='class 1')
plt.scatter(X[50:,0],X[50:,1] ,c='blue',label='class 2')
x1=np.linspace(-20, 20 ,200)
b=0.7
x4=-(Wf[0]/Wf[1])*x1 +b
plt.plot(x1,x4,c='orange',label='Fisher')
plt.title('Fisher')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()
plt.show()

#fisher_error
z=Wf[1]*X[:,1]+Wf[0]*X[:,0]+b
result = np.where(z >= 0, 1, -1)
label=np.reshape(label, (result.shape[0],))
differ=result==label
true_numbers=np.count_nonzero(differ)
accuracy=(true_numbers/label.shape[0])*100
print('accuracy of fisher=',accuracy)
```

```
 fisher separating line: 0.33553874336894723 * x + -0.942026407113085 * y = 0
[[0.12424898 0.01613878]
 [0.01613878 0.03010612]]
```

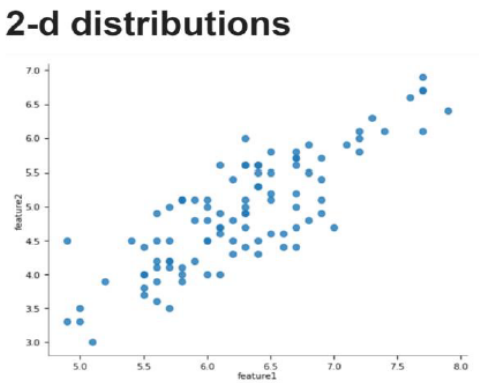

```
 fisher separating line: 0.33553874336894723 * x + -0.942026407113085 * y = 0
[[0.12424898 0.01613878]
 [0.01613878 0.03010612]]
```


```
accuracy of fisher= 100.0
```

Answer:

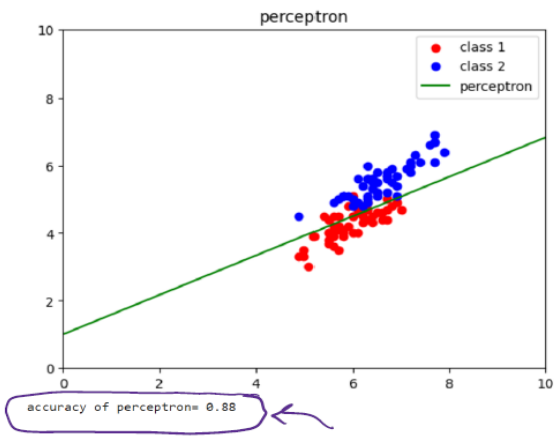
As we see in this case the data distribution is no longer linearly separatable in this case the perceptron model undoubtedly has error.



part B_1 (perceptron)

```
#Error
z=final_weights[2]*X[:,1]+final_weights[1]*X[:,0]+final_weights[0]
result = np.where(z >= 0, 1, -1)
label=np.reshape(label, (result.shape[0],))
differ=result==label
true_numbers=np.count_nonzero(differ)
accuracy=true_numbers/label.shape[0]
print('accuracy of perceptron=',accuracy)
```

```
weights= [ 222.00417908  128.7539688  -220.9866857 ]     numbers of update= 5337
```

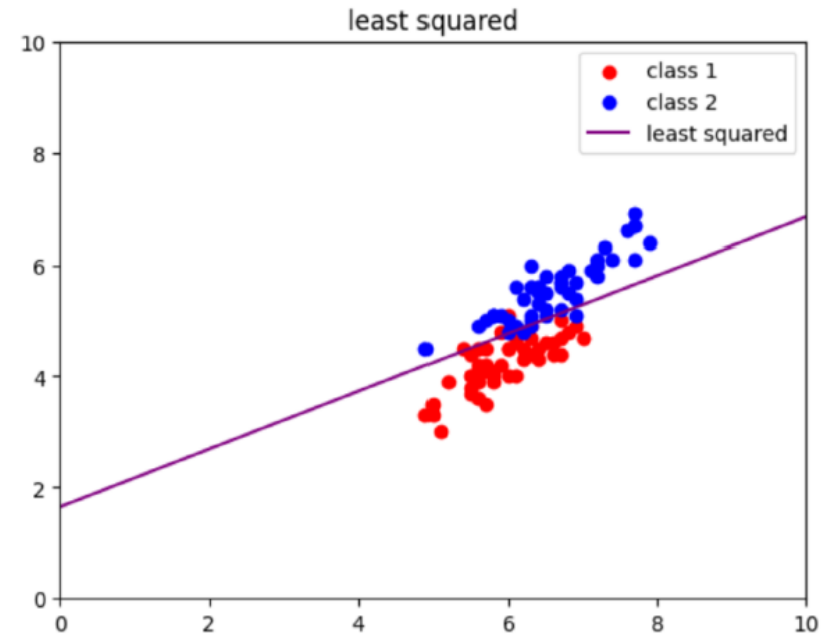


part B_2 (Least square)

Least square works better in this case because it uses minimization and the project the target vector in the answers subspace and hence it can reach to a better answer than perceptron.

```
#accuracy
z=W[2]*X[:,1]+W[1]*X[:,0]+W[0]
result = np.where(z >= 0, 1, -1)
label=np.reshape(label, (result.shape[0],))
differ=result==label
true_numbers=np.count_nonzero(differ)
accuracy=true_numbers/label.shape[0]
print('accuracy of least square=',accuracy)
```

```
least squared weights= [ 2.41736342  0.76016085 -1.46300258]
```



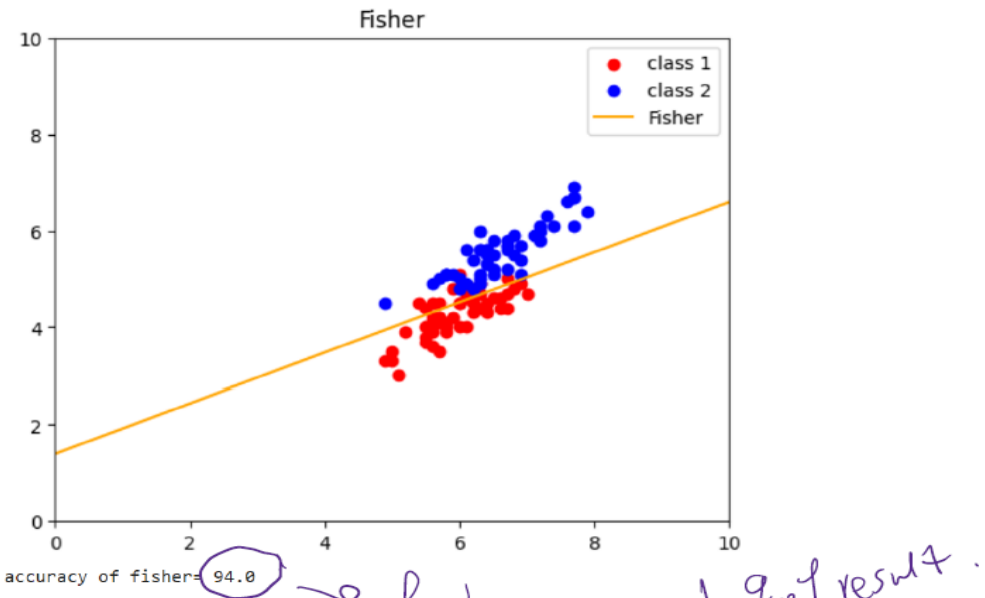

```
accuracy of least square= 0.94
```

part B_3 (Fisher)

with considering the appropriate bias fisher can yield best result.

```
#fisher_error
z=Wf[1]*X[:,1]+Wf[0]*X[:,0]+b
result = np.where(z >= 0, 1, -1)
label=np.reshape(label, (result.shape[0],))
differ=result==label
true_numbers=np.count_nonzero(differ)
accuracy23=(true_numbers/label.shape[0])*100
print('accuracy of fisher=',accuracy23)
```

```
 fisher separating line: 0.46106601461769925 * x + -0.8873658378394739 * y + 1.4 = 0
[[0.26643265 0.18289796]
 [0.18289796 0.22081633]]
```



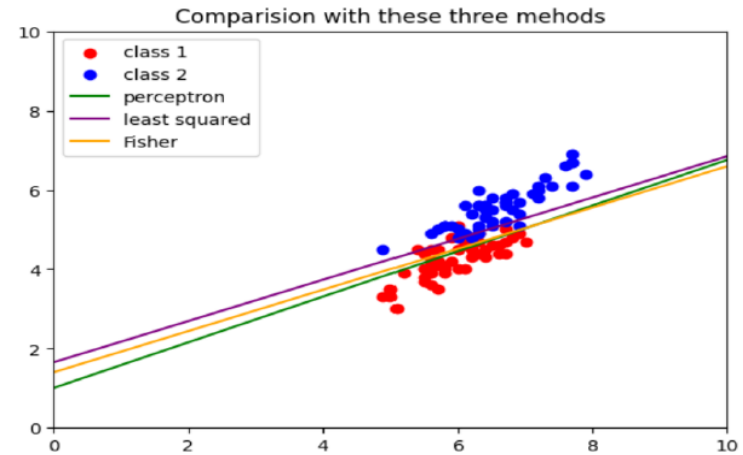

```
accuracy of fisher= 94.0
```

→ fisher can reach good result.

Quite similar to the response of least squared method (by choosing best bias).

part B_4 (comparison)

```
plt.scatter(X[0:50,0],X[0:50,1] ,c='red',label='class 1')
plt.scatter(X[50:,0],X[50:,1] ,c='blue',label='class 2')
plt.plot(x1,x2,c='green',label='perceptron')
plt.plot(x1,x3,c='purple',label='least squared')
plt.plot(x1,x4,c='orange',label='Fisher')

plt.title('Comparision with these three mehods')
plt.xlim(0,10)
plt.ylim(0,10)
plt.legend()
plt.show()
print('accuracy of perceptron=',accuracy21)
print('accuracy of least square=',accuracy22)
print('accuracy of fisher=',accuracy23)
```




```
accuracy of perceptron= 0.86
accuracy of least square= 0.94
accuracy of fisher= 94.0
```

The error of **perceptron** in this case is pretty high because our data is no longer linearly separatable and it certainly has error. This error is heavily depending upon the initialization weights and numbers of epochs. **Least square** works better in this case because it uses minimization and the project the target vector in the answers subspace and hence it can reach to a better answer than perceptron. And **Fisher** also can reach to similar conclusion with considering good bias and it completely fails to predict but comprehensively it fails when data distribution is not gaussian anymore and our data cannot be separate with lower dimension and if we don't consider a good separation bias.

part C

For comparison between part A and B, is explained completely in the previous parts. Distribution of part A is linearly separatable so **perceptron** in this case guarantees to find a separator without error (giving required iterations) and its performance is heavily depends upon initialization weights and numbers of iteration so it can't get good result in part B. While **least square** which use minimization works acceptable in both case in the second case has the minimum error because it tries to find closest answer. Finally, **Fishers** needs a gaussian distribution (to be compressable) and we have to tune it's bias but if we provide good tune for the bias it can reach the comparable result.

part D

here we use the logistic regression which can find better discriminator w.r. t former cases. In this section we used the ML library SKlearn as allowed in the question(instead of writing it from the scratch).

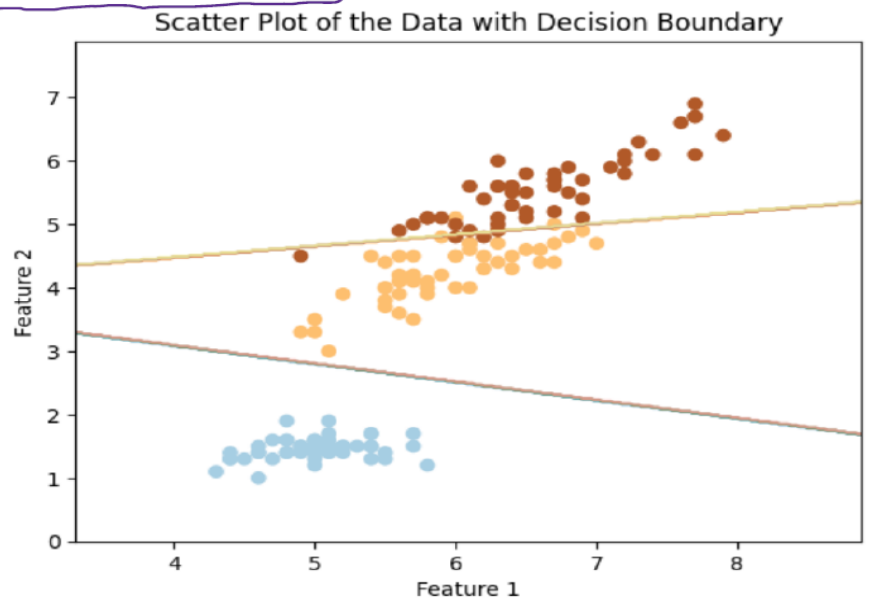
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# we dont use trining set and test set split because we have not large data set and we test our algorithm in the training set.
#because we just want seperator.
X_train, X_test, label_train, label_test = train_test_split(X, label, test_size=0.2, random_state=42)

label_train = label_train.ravel()
label_test = label_test.ravel()

logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X, label.ravel())

logistic_predictions = logistic_model.predict(X)

correct_predictions = np.sum(logistic_predictions == label.ravel())
total_predictions = len(label.ravel())
percentage_accuracy = (correct_predictions / total_predictions) * 100

print('Correct Predictions:', correct_predictions)
print('Total Predictions:', total_predictions)
print('Percentage Accuracy:', percentage_accuracy)

# Visualization
plt.scatter(X[:, 0], X[:, 1], c=label, cmap=plt.cm.Paired)
plt.title('Scatter Plot of the Data with Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the decision boundary
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
k=np.arange(x_min, x_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logistic_model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.show()

print('weights of the dicision boundries: \n',logistic_model.coef_)
```

Here is the result.

```
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
k=np.arange(x_min, x_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logistic_model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.show()

print('weights of the dicision boundries: \n',logistic_model.coef_)
```

```
Correct Predictions: 144
Total Predictions: 150
Percentage Accuracy: 96.0
```

← accuracy




```
weights of the dicision boundries:
 [[-0.27572275 -3.10597407]
 [ 0.49116052 -0.43375975]
 [-0.21543778  3.53973382]]
```

As we see logistic regression has the best accuracy with respect to the former classifiers which provided in the previous sections and it just misclassifies 6 labels and it reached the accuracy of 96% which is the total accuracy which is maximum. And for the non-discriminative class 2 and

three the accuracy is 94% which is also the maximum. And for the linearly separatable classes 0 and 1 it predicts all the classes correctly and provided 100% accuracy. So linear regression provided the best accuracy with comparison to previous approaches(similar to least squared in this particular case) and it can also extended easily to multiclass cases.

```
Correct Predictions: 144
Total Predictions: 150
Percentage Accuracy: 96.0
```

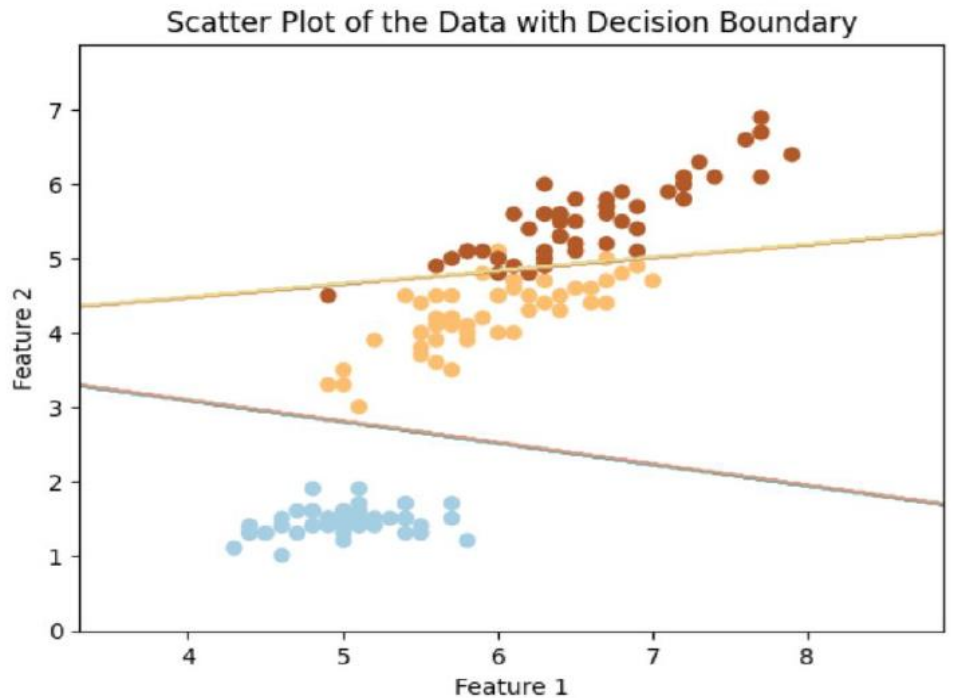
Total accuracy for all classes

```
Correct Predictions for class 1 and 2: 94
Total Predictions: for class 1 and 2 100
Percentage Accuracy for class 1 and 2: 94.0
```

} class 2 & 1 accuracy

```
Correct Predictions for class 0 and 1: 100
Total Predictions: for class 0 and 1 100
Percentage Accuracy for class 0 and 1: 100.0
```

} class 0 & 1 accuracy




```
weights of the dicision boundries:
 [[-0.27572275 -3.10597407]
 [ 0.49116052 -0.43375975]
 [-0.21543778  3.53973382]]
(100,)
```

Part A (classes 0 and 1)

Logistic regression	perceptron	Least squared	Fisher
100%	100%	100%	100%

Part B (classes 1 and 2)

Logistic regression	perceptron	Least squared	Fisher
94%	88%	94%	94%

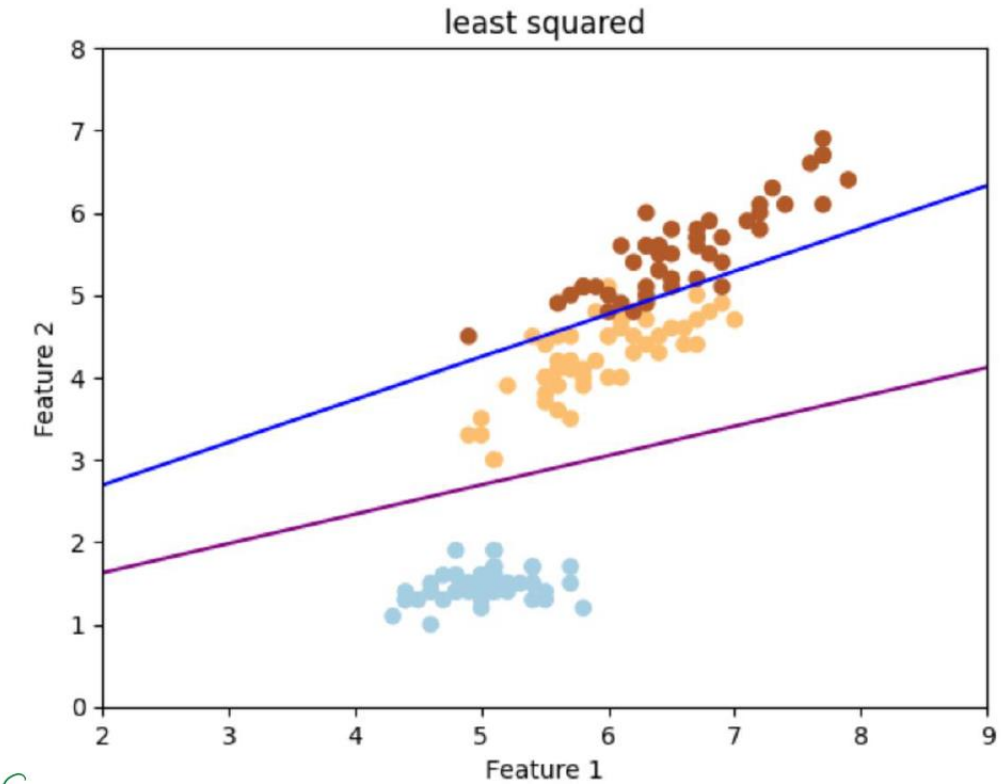
part E

here we use one against one algorithm for classifying with least square method.

```
plt.scatter(X[:, 0], X[:, 1], c=label, cmap=plt.cm.Paired)
plt.title('Scatter Plot of the Data with Decision Boundary Least square')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')


b=np.ones((100,1))
W=np.linalg.inv((Y1.T@Y1))@Y1.T@b
print('least squared weights=',W)
w=[-W[0]/W[2] ,-W[1]/W[2]]

W2=np.linalg.inv((Y2.T@Y2))@Y2.T@b
print('least squared weights=',W2)
w2=[-W2[0]/W2[2] ,-W2[1]/W2[2]]


x1=np.linspace(-20, 20 ,200)

# Visualization
x3=w[1]*x1+w[0]
x4=w2[1]*x1+w2[0]

plt.plot(x1,x3,c='purple')
plt.plot(x1,x4,c='blue')

plt.title('least squared')
plt.xlim(2,9)
plt.ylim(0,8)
plt.show()
```

```
least squared weights= [[ 0.70498922]
 [ 0.27494856]
 [-0.7719192 ]]
least squared weights= [[ 2.41736342]
 [ 0.76016085]
 [-1.46300258]]
```

coefficients




```
accuracy of least square class 0 & 1= 100.0 %
accuracy of least square for class 1 & 2 = 94.0 %
Total accuracy = 96.0 %
Total number of misclassified = 6
```

accuracy

Thank you!

Feel free to contact me

Mehrant.0611@gmail.com
Telegram: @Mttnt