SMAI Assignment 2

Roll No – 20162080
Name – Kanishtha Surana
Branch – Mtech Cse

Problem 1: Perceptron Learning

Date Set

samples	C_1		C_2		C_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

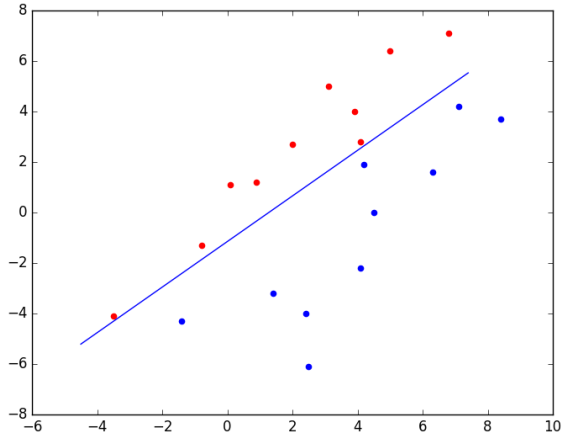
Table 1: Data for Problem 1

- **Initialization :** a = [0,0,0] , learning_rate =0.1
- **Code Snippet**

```
def online_perceptron(data,a,learning_rate):
        k=0
        flg = False
        while flg == False:
                k=k+1
                flg = True
                for i in xrange(len(data)):
                        temp = numpy.inner(a,data[i])
                        if temp <= 0 :
                                delta = numpy.dot(data[i],learning_rate)
                                a = numpy.add(a,delta)
                                flg = False
        print "iterations = "+str(k)
        return a
```

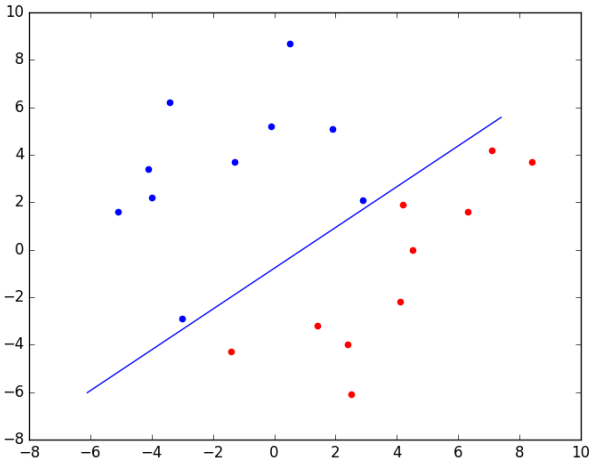
- **OUTPUT**
- **For Class 1 and Class 2**

```
kanishtha@kanishtha-Vostro-3558:~/Desktop/Smai_Assignment2/q1$ python online_perceptron.py class1.csv class2.csv
iterations = 9
[-1.02  1.13  1.3 ]
```



- **For Class 2 and Class 3**

```
kanishtha@kanishtha-Vostro-3558:~/Desktop/Smai_Assignment2/q1$ python online_perceptron.py class2.csv class3.csv
iterations = 5
[ 0.55 -0.64 -0.5 ]
```



3. **Comment on the difference on the number of iteration required for convergence in above two cases and give reasons.**

No of iterations required in the first case is 9 iterations whereas in the second case only 5 iterations are required while the learning rate is both in the same case. The main reason for the difference between number of iterations is because of the data points. In first case the data points are close and that why weight vector(if the point is misclassified) is updated by a small margin and so moving the weight vector in the solution region takes more time as compared to second case where the data is scattered far away and so the weight vector is updated by more margin.

Problem 2: Voted Perceptron

Data Set:

• Breast Cancer Dataset: http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/
• Ionosphere Dataset: http://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/

Algo -

Algorithm 1: Voted Perceptron

Input: $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$
Output: $(\mathbf{w}_1, b_1, c_1), \ldots, (\mathbf{w}_n, b_n, c_n)$
begin
  Initialization $n = 1,\ \mathbf{w}_1 = \mathbf{0}, b_1 = 0, c_1 = 1;$
  for $iter \leftarrow 1$ **to** Num_Epochs **do**
    for $i \leftarrow 1$ **to** m **do**
      if $y_i(\mathbf{w}_n^T \mathbf{x}_i + b_n) \leq 0$ **then**
        $n = n + 1;$
        $\mathbf{w}_n = \mathbf{w}_{n-1} + y_i \mathbf{x}_i;$
        $b_n = b_{n-1} + y_i;$
        $c_n = 1$
      end
      $c_n = c_n + 1$
    end
  end
end

Code Snippet -

- **Voted Perceptron -**

```
def voted_perceptron(data_classlabels,dimension,epoch):
        tmp_wt=np.zeros(dimension)
        w=np.empty((0,2))
        count=1
        for i in range(epoch):
                for row in data_classlabels:
                        feature_vector=row[0]
                        feature_vector=np.array(feature_vector)[np.newaxis]
                        feature_vector=feature_vector.transpose()
                        feature_vector=np.matrix(feature_vector)
                        if np.dot(tmp_wt,feature_vector)<=0:
                                w=np.append(w,create_2d_array(tmp_wt,count),axis=0)
                                tmp_wt=tmp_wt+feature_vector.transpose()
                                count=1
                        else:
                                count=count+1

        return w
```

- **Vanilla Perceptron-**

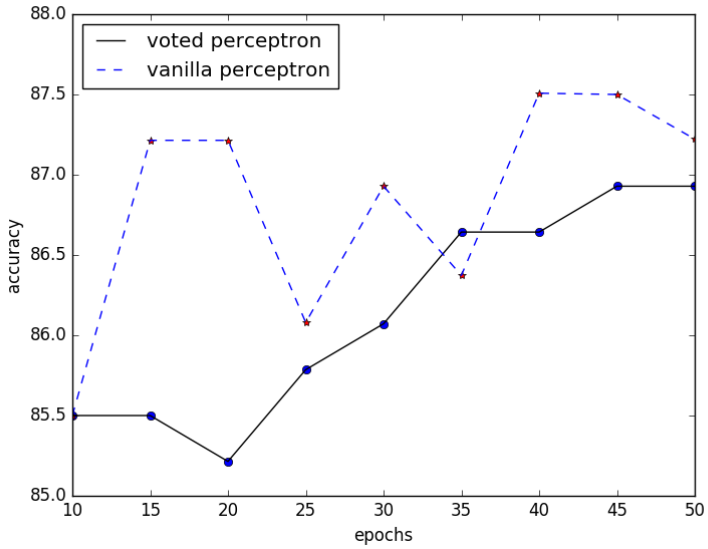
```
def vanilla_perceptron(samples,dimension,epoch):
```

```
w=np.zeros(dimension)
for i in range(epoch):
        for row in samples:
                feature_vector=row[0]
                feature_vector=np.array(feature_vector)[np.newaxis]
                feature_vector=feature_vector.transpose()
                feature_vector=np.matrix(feature_vector)
                if np.dot(w,feature_vector)<=0:
                        w=w+feature_vector.transpose()
#print w.shape
return w
```

Output -

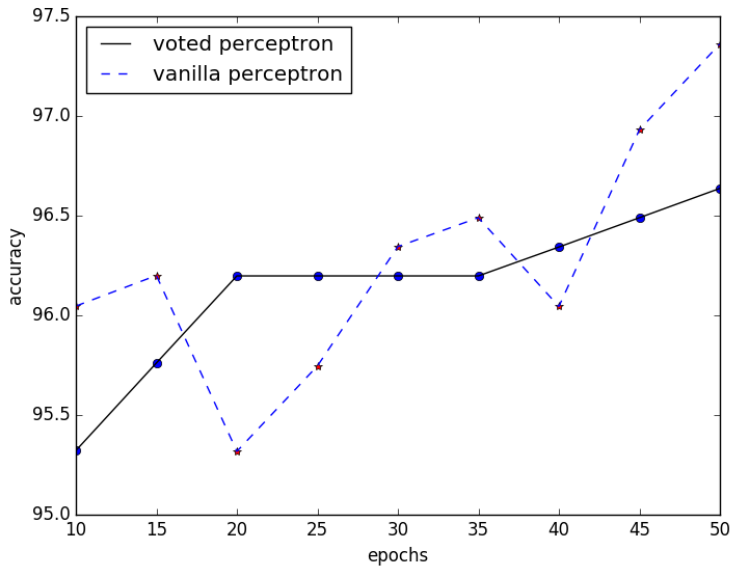
For Ionospere data set

```
[85.49999999999999, 85.49999999999999, 85.21428571428571, 85.78571428571428, 86.07142857142856, 86.64285714285714, 8
6.64285714285714, 86.92857142857143, 86.92857142857143]
[85.5, 87.21428571428572, 87.21428571428572, 86.07936507936508, 86.92857142857143, 86.37301587301587, 87.50793650793
652, 87.49999999999999, 87.22222222222223]
```



For Breast-Cancer data set -

```
Accuracies-Voted-- [95.3218243819267, 95.76086956521738, 96.19778346121058, 96.19778346121058, 96.19778346121058, 96
.19778346121058, 96.34271099744247, 96.48976982097187, 96.63682864450126]
Accuracies-Vanilla-- [96.0464620630861, 96.19991474850809, 95.31756180733166, 95.7459505541347, 96.34484228474, 96.4
8976982097189, 96.04433077578858, 96.9309462915601, 97.3614663256607]
kanishtha@kanishtha-Vostro-3558:~/Desktop/Smai_Assignment2$
```



Comment on the performance of two approaches. Give

reasons.

Advantage of Voted Perceptron-

In case of Voted Perceptron , the vote count has been taken under consideration , the weight vector which classifies more number of sample properly gets higher significance in deciding the final prediction. Thus, from epoch-accuracy (obtained using 10-fold cross validation) graph we see, for a set of data, the Voted Perceptron saturates after a certain value of epoch, while accuracy of Vanilla Perceptron fluctuates which does not guarantee better performance .

Disadvantage of voted Perceptron -

Voted Perceptron requires greater storage space. The voting perceptron stores every single weight vector computed.This takes $O(T * |W|)$ space to store where T is the number of iterations we train and $|W|$ is the size of the weight vector. This can be huge for many normal problems as opposed to the averaged perceptron which only require $O(|W|)$ space to store its weight vector.

Problem 3: Least Square Approach

Data Set

samples	C_1		C_2	
	x_1	x_2	x_1	x_2
1	3	3	-1	1
2	3	0	0	0
3	2	1	-1	-1
4	0	2	1	0

Table 2: Dataset 1 for Problem 3

samples	C_1		C_2	
	x_1	x_2	x_1	x_2
1	3	3	-1	1
2	3	0	0	0
3	2	1	-1	-1
4	0	1.5	1	0

Table 3: Dataset 2 for Problem 3

Part 1. Write the program to find the linear classifier using least square approach.

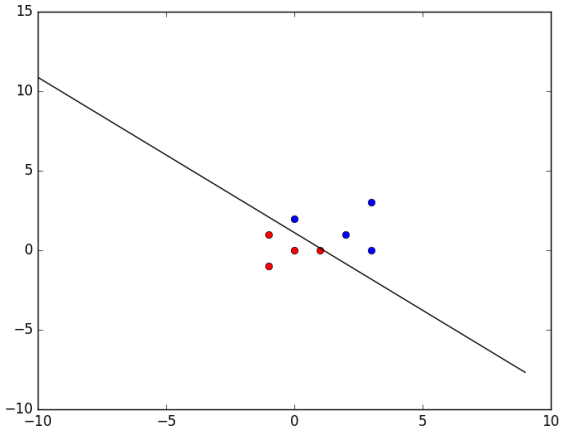
Code Snippet:

```
def lms(data,a,learning_rate,b):
        e=0.0001
        k=1
        n=len(data)
        a+= learning_rate * (b-np.dot(a,data[0]))*array(data[0])/k
        step=0
        while step <1000:
                if k==0:
                        k=1
                a+= learning_rate * (b-np.dot(a,data[k]))*array(data[k])/k
                if norm(b-np.dot(a,data[k])) < e:
                        break
                step+=1
                k=(k+1)%n
        print a
        return a
```

Output -

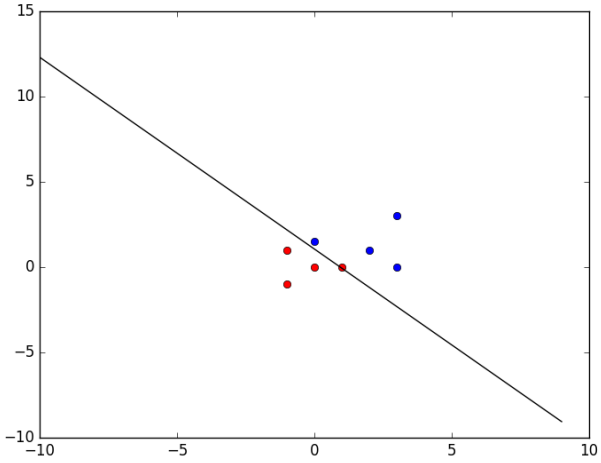
- **Data Set1 :**

```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python lms.py c1a.csv c1b.csv
[ 0.04976174  0.05094654 -0.05670911]
```



- **Data Set 2**

```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python lms.py c2a.csv c2b.csv
[ 0.0531201   0.04723801 -0.05019429]
```



2. Write the program to find the linear classifier using Fisher's linear discriminant.

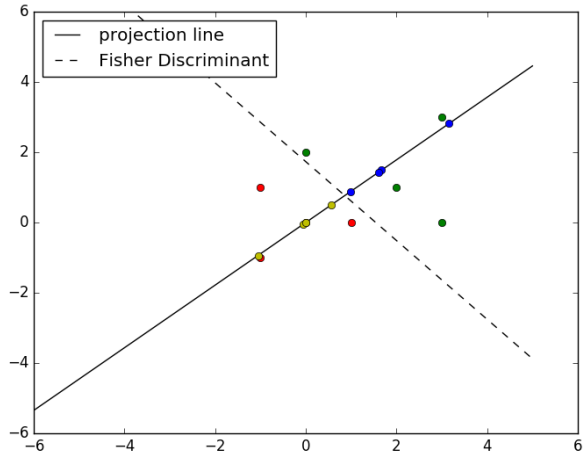
Code Snippet -

```
c1,c2 = generateData(fname1,fname2)
        m1=np.mean(c1, axis=0)
        m2=np.mean(c2, axis=0)
        Sw=np.dot((c1-m1).T, (c1-m1))+np.dot((c2-m2).T, (c2-m2))
        w=np.dot(np.linalg.inv(Sw), (m2-m1))
        print "weight vector--", w
```

Output -

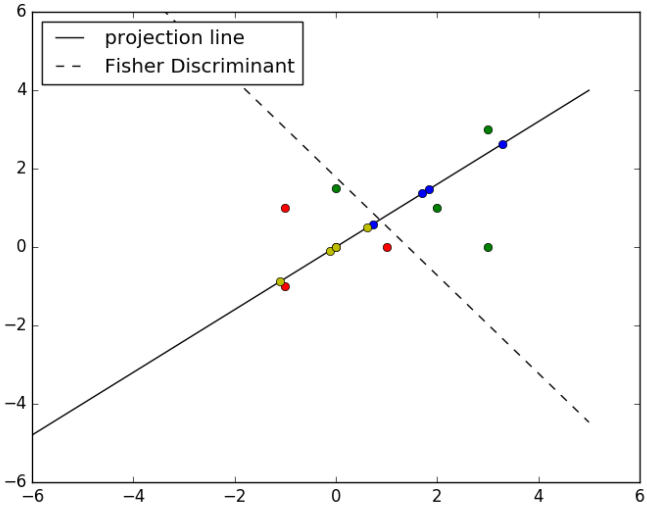
- **Data Set1 :**

```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python fisher_imp.py c1a.csv c1b.csv
[[3.0, 3.0], [3.0, 0.0], [2.0, 1.0], [0.0, 2.0]]
[[-1.0, 1.0], [0.0, 0.0], [-1.0, -1.0], [1.0, 0.0]]
weight vector-- [-0.28630705 -0.25518672]
```



- **Data Set 2**

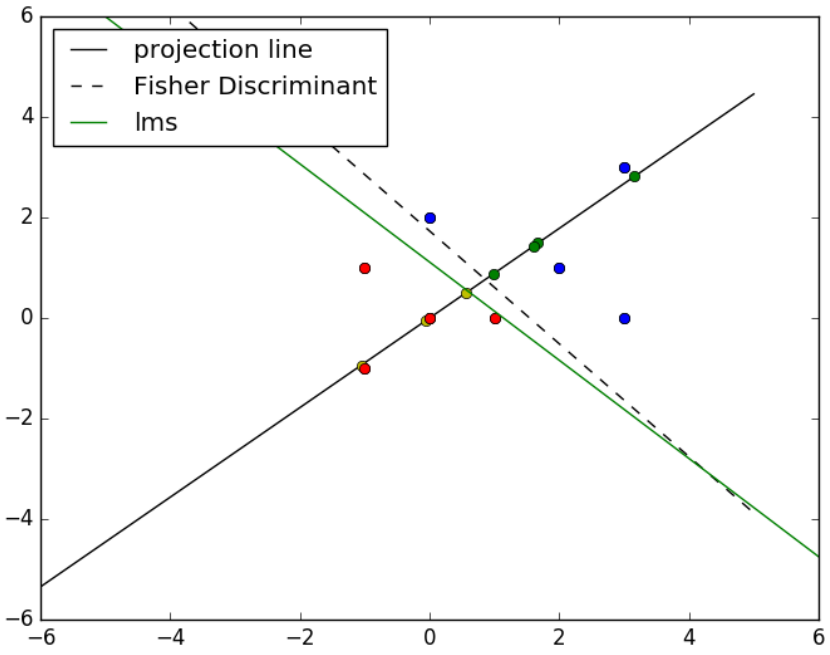
```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python fisher_imp.py c2a.csv c2b.csv
[[3.0, 3.0], [3.0, 0.0], [2.0, 1.0], [0.0, 1.5]]
[[-1.0, 1.0], [0.0, 0.0], [-1.0, -1.0], [1.0, 0.0]]
weight vector-- [-0.25714286 -0.20560748]
```



3. Plot the data points in Table 2 with different colors for di↩erent classes. Find the classifiers learnt using both least square approach as well as Fisher's linear discriminant analysis. Plot both the classifiers on the same graph.

Output-

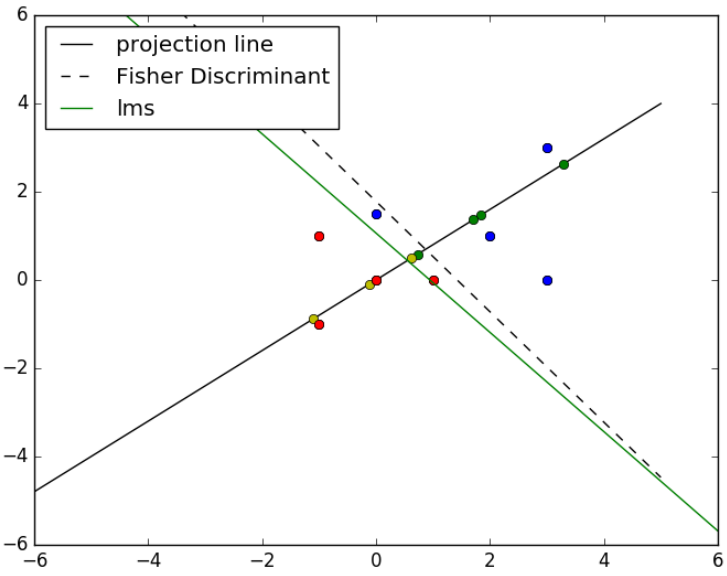
```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python fisher_lms.py c1a.csv c1b.csv
[[3.0, 3.0], [3.0, 0.0], [2.0, 1.0], [0.0, 2.0]]
[[-1.0, 1.0], [0.0, 0.0], [-1.0, -1.0], [1.0, 0.0]]
Weight vector in case of Fisher-- [-0.28630705 -0.25518672]
Weight Vector in case of LMS-- [ 0.04976174  0.05094654 -0.05670911]
```



4. Plot the data points in Table 3 with different colors for di↩erent classes. Find the classifiers learnt using both least square approach as well as Fisher's linear discriminant analysis. Plot both the classifiers on the same graph.

Output -

```
kanishtha@kanishtha-Vostro-3558:~/Desktop$ python fisher_lms.py c2a.csv c2b.csv
[[3.0, 3.0], [3.0, 0.0], [2.0, 1.0], [0.0, 1.5]]
[[-1.0, 1.0], [0.0, 0.0], [-1.0, -1.0], [1.0, 0.0]]
Weight vector in case of Fisher-- [-0.25714286 -0.20560748]
Weight Vector in case of LMS-- [ 0.0531201   0.04723801 -0.05019429]
```



5. Comment on the difference in the classifier learnt in the above two cases. Give reasons.

LMS Classifier

The basic idea behind LMS filter is to approach the optimum filter weights , by updating the filter weights in a manner to converge to the optimum filter weight. The algorithm starts by assuming initial weight vector and, at each step, by finding the gradient of the mean square error, the weights are updated.The disadvantage of Lms is even if the data is linearly separable,it does not result in a classifier that linearly separates the feature vectors.

Fisher's Classifier

Fisher's LDA transformation will allow us to reduce the dimensions of the data and then apply some classification algorithm to find decision boundary.

Problem 4: Relation between Least Squares and Fisher' linear discriminant

Assignment 2

Q14: Let there be n_1 ~~pot~~ feature vectors (d dimension) belonging to class ω_1

$x_1, x_2 \cdots x_{n_1} \in \omega_1$

n_2 feature vectors belonging to class ω_2

$x_{n_1+1} \cdots x_{n_1+n_2} \in \omega_2$

if $x_i \in \omega_2$ then negate x_i for normalization

Now $$Y = \begin{bmatrix} I_1 & X_1 \\ -I_2 & -X_2 \end{bmatrix}$$

Here $$I_1 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n_1 \times 1} \qquad I_2 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n_2 \times 1}$$

I_i : Column vector of n_i ones

X_i : an $n_i \times d$ matrix

$$a = \begin{bmatrix} \omega_0 \\ \omega \end{bmatrix} \qquad b = \begin{bmatrix} \frac{n}{n_1} I_1 \\ \frac{n}{n_2} I_2 \end{bmatrix}$$

In MSE –

Error function $\Rightarrow$ $e = Ya - b$

$$J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} (a^t y_i - b_i)^2$$

$$\nabla_a J_s = \sum_{i=1}^{n} 2(a^t y_i - b_i) y_i$$

$$= 2Y^t(Ya - b)$$

Now for closed form solution

$\nabla_a J_s = 0$

$$\therefore \quad Y^t Y a = Y^t b \quad \text{——} \quad (1)$$

Now placing the value of Y^t, a and b in eq^n (1)

$$\begin{bmatrix} I_1^t & -I_2^t \\ X_1^t & -X_2^t \end{bmatrix} \begin{bmatrix} I_1 & X_1 \\ -I_2 & -X_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w \end{bmatrix} = \begin{bmatrix} I_1^t & -I_2^t \\ X_1^t & -X_2^t \end{bmatrix} \begin{bmatrix} \frac{n}{n_1} I_1 \\ \frac{n}{n_2} I_2 \end{bmatrix}$$

—— (2)

In case of Fisher LDA –

$$m_i = \frac{1}{n_i} \sum_{x \in D_i} x \quad \Rightarrow \text{means}$$

$$S_w = \sum_{i=1}^{2} \sum_{x \in D_i} (x - m_i)(x - m_i)^t$$

Now solving (2)

$$\begin{bmatrix} n_1 + n_2 & (n_1 m_1 + n_2 m_2)^t \\ n_1 m_1 + n_2 m_2 & S_w + n_1 m_1 m_1^t + n_2 m_2 m_2^t \end{bmatrix} \begin{bmatrix} w_0 \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ n(m_1 - m_2) \end{bmatrix}$$

$n_1 + n_2 = n$.

$$\frac{n w_0 + w(n_1 m_1 + n_2 m_2)^t}{n} = 0 \quad \text{---} \quad (3)$$

$$w_0 + w m^t = 0$$

$$\therefore \boxed{w_0 = -m^t w}$$

$\left\{ m = \frac{m_1 n_1 + m_2 n_2}{n} \right.$ ↳ overall mean $\left. \right\}$

Now solving the 2nd eqⁿ and substituting $w_0 = -m^t w$.

$$\left[\frac{1}{n} S_w + \frac{n_1 n_2}{n^2} (m_1 - m_2)(m_1 - m_2)^t \right] w = m_1 - m_2$$

The vector $(m_1 - m_2)(m_1 - m_2)^t w$ is in the direction of $(m_1 - m_2)$ for any value of w.

$$\frac{n_1 n_2}{n^2} (m_1 - m_2)(m_1 - m_2)^t w = (1 - \alpha)(m_1 - m_2)$$

$\alpha \rightarrow$ some scalar.

$$\frac{S_w}{n} w = m_1 - m_2$$

$$w = \underbrace{\alpha n} S_w^{-1} (m_1 - m_2)$$

↳ unimportant scale factor.

Apart from this scale factor of αn this solⁿ is same as Fisher's linear discriminant.