BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

HYDERABAD CAMPUS

BITS F464 MACHINE LEARNING

ASSIGNMENT - I

Donkada Vishal Dheeraj	2018A7PS0239H
Thakur Shivank Singh	2018A7PS0439H
Pranav Reddy Pesaladinne	2018A7PS0238H

Fisher's Linear Discriminant

Description of Algorithm

Fisher's linear discriminant is a classification algorithm that solves the problem of discriminating the data points by reducing the cluster into 1D from the original dimension. The main idea is to find a unit vector (*w*) that when all the data points are projected onto it makes it easier for us to find the discriminant.

Implementation

To get the best possible result the unit vector (*w*) should maximize the difference between the means (m_1) and (m_2) of the positive and negative points respectively when projected onto *w*. This makes sure that the positive and negative reduced clusters are as far away from each other as possible.

$$\text{Maximize } (m_1 - m_2)^2$$

Simplifying this we can conclude that

$$\mathbf{w} \propto \mathbf{(M_1 - M_2)}$$

where M_1 and M_2 are means of positive and negative points in the original space

Now, to further improve our result we have to make sure that the variances of the positive and negative data points do not increase abnormally after projection. Let's say $(s_1)^2$ and $(s_2)^2$ are the variances of positive and negative data after projection, to attain the above optimization we should minimize their sum

$$\text{Minimize}(s_1^2 + s_2^2)$$

We combine both the constraints to arrive at

$$\text{Maximize } [(m_1 - m_2)^2 / (s_1^2 + s_2^2)]$$

On further simplifications, the equation boils down to

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

$$\Rightarrow \mathbf{w} \propto \mathbf{S_W^{-1} (M_2 - M_1)}$$

Where $S_W = \sum_{n \in C1} (x_n - M_1)(x_n - M_1)^T + \sum_{n \in C2} (x_n - M_2)(x_n - M_2)^T$:

$C_1 \rightarrow$positive class , $C_2 \rightarrow$ negative class

Means M1 and M2 in original Space

```
#calculate means in original space
M1 = np.mean(positive, axis = 0)
M2 = np.mean(negative, axis = 0)
```

```
print("M1 =",M1)
print("M2 =",M2)
```

```
M1 = [0.22310309 0.00255859 1.01027072]
M2 = [ 0.10229483  0.10337021 -1.00513   ]
```

S_w^{-1} and w Calculation

```
#Calculating Sw
res1 = np.zeros([3,3])
for i in range(len(positive)):
    pos = positive[i]-M1
    pos.shape = (1,3)
    posT = np.transpose(pos)
    ans1 = np.dot(posT,pos)
    res1 = np.add(res1,ans1)
```

```
res2 = np.zeros([3,3])
for i in range(len(negative)):
    neg = negative[i]-M2
    neg.shape = (1,3)
    negT = np.transpose(neg)
    ans2 = np.dot(negT,neg)
    res2 = np.add(res2,ans2)
```

```
res1 = res1/len(positive)
res2 = res2/len(negative)
```

```
Sw = res1+res2
print("Sw is\n",Sw)
```

```
#Calculating Sw inverse
sw_inv = np.linalg.inv(Sw)
sw_inv
```

```
array([[ 0.05552224,  0.00607198, -0.03985285],
       [ 0.00607198,  0.50861384, -0.07735819],
       [-0.03985285, -0.07735819,  5.61424388]])
```

```
#Finding the vector w and normalising it
w = np.dot(sw_inv,np.transpose(M2-M1))
print("w is",w)
```

```
w is [  0.07422404   0.20644839 -11.31793517]
```

```
import math
mag = math.sqrt(w[0]*w[0]+w[1]*w[1]+w[2]*w[2])
```

```
w = w/mag
print("normalised w is",w)
```

```
normalised w is [ 0.00655686  0.01823739 -0.99981218]
```

```
#Fitting the reduced clusters into Gauss Normal Distributions
mu_p = np.mean(positive_projections)
std_p = np.std(positive_projections)
mu_n = np.mean(negative_projections)
std_n = np.std(negative_projections)
```

Fitting reduced data into Normal Distributions

```
def findThreshold(mu1,mu2,std1,std2):
    p = 1/(2*std1**2) - 1/(2*std2**2)
    q = mu2/(std2**2) - mu1/(std1**2)
    r = mu1**2 /(2*std1**2) - mu2**2 / (2*std2**2) - np.log(std2/std1)
    return np.roots([p,q,r])
```

Finding the roots of both the Normal Distributions

```
intersection = findThreshold(mu_p,mu_n,std_p,std_n)
threshold = 0
for i in range(len(intersection)):
    if (mu_p < intersection[i]) and (mu_n > intersection[i]):
        threshold = intersection[i]
        break

print("Threshold is",threshold)
```

Threshold is a root which lies between the means (μp , μn) of reduced positive and negative clusters

Threshold = 0.3893028020993765

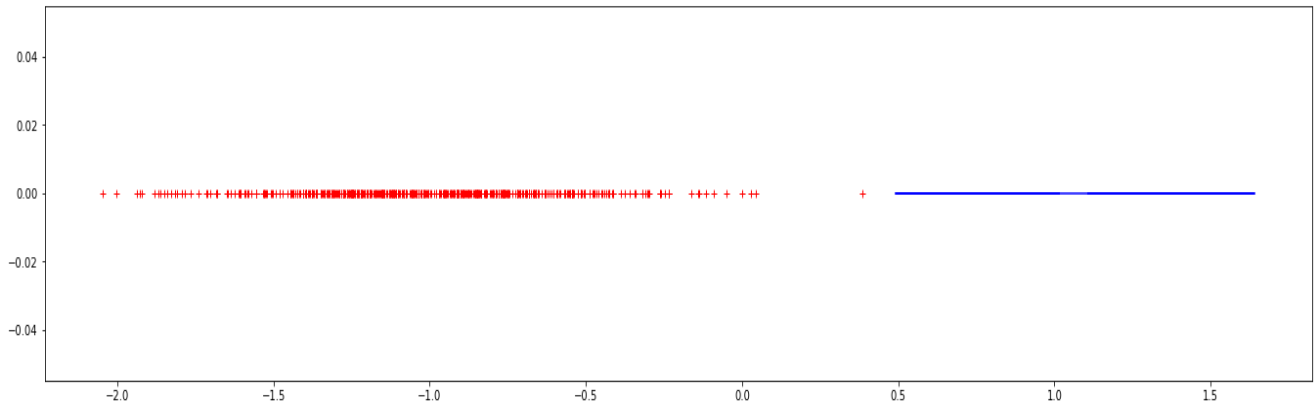
Testing Accuracy:

```
correct_pclass = 0
for i in range(len(positive)):
    if np.dot(w,np.transpose(positive[i])) < threshold:
        correct_pclass += 1

correct_nclass = 0
for i in range(len(negative)):
    if np.dot(w,np.transpose(negative[i])) > threshold:
        correct_nclass += 1
accuracy = (correct_pclass + correct_nclass)/(len(positive) + len(negative))
print("Accuracy is",accuracy)
```

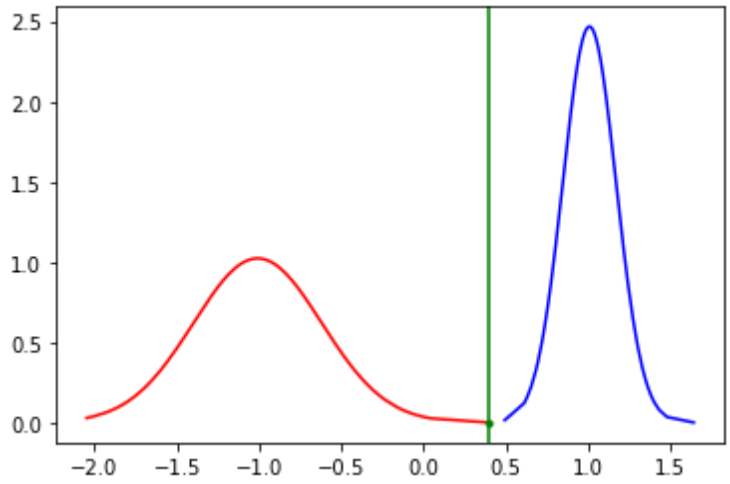
```
Accuracy is 1.0
```

Reduced Cluster (Positive : + and Negative : -)



Upon arriving at our required vector *w,* we have to determine the threshold value that classifies the projected positive and negative points in 1D. We do this by fitting both the positive and negative clusters into two Gauss Normal distributions appropriately and conclude the intersection of the two curves as our threshold (*b*). The line perpendicular to *w* passing through *b* is the discriminant line in one dimension.

Gauss Normal distributions of Positive and Negative Projections on *w*



Red Curve : Normal Distribution of +
Blue Curve : Normal Distribution of -

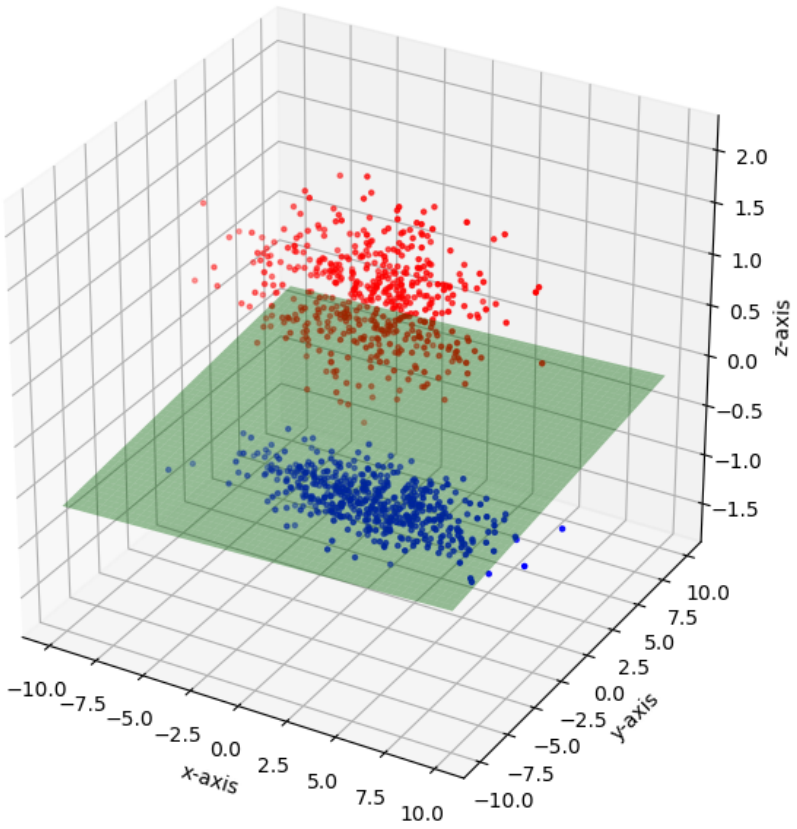
Green Line : Discriminant in 1D

The discriminant line in the original dimension is determined by

$$\mathbf{w^T x = b}$$

where b is the threshold value

Data Points and Discriminant in Original Space



Red Points Belong to the Positive Class
Blue Points Belong to the Negative Class
Green Plane denotes the Discriminant in Original Space

Final Results

```
#Printing Final results
print("Unit vector w :",w)
print("Threshold in 1D =",threshold)
print("Accuracy :",accuracy)
```

```
Unit vector w : [ 0.00655686  0.01823739 -0.99981218]
Threshold in 1D = 0.3893028020993765
Accuracy : 1.0
```

Packages used : Pandas, Numpy, Matplotlib, scipy.norm