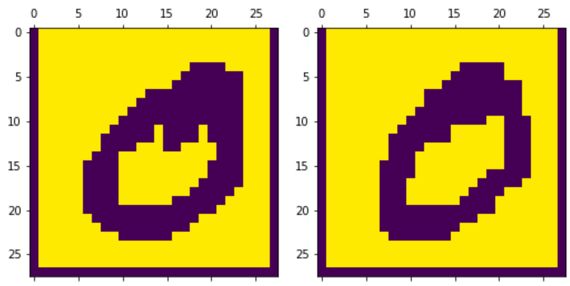
Elon Brange, 埃隆
0645120

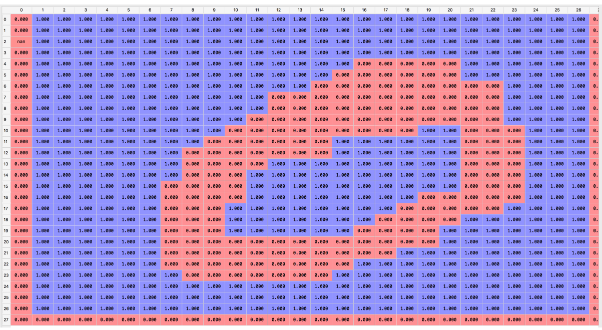
Linear Discriminant Analysis (LDA, Fishers discriminant analysis) –Extra Homework, Machine Learning 5255,9i NCTU

Dataset:

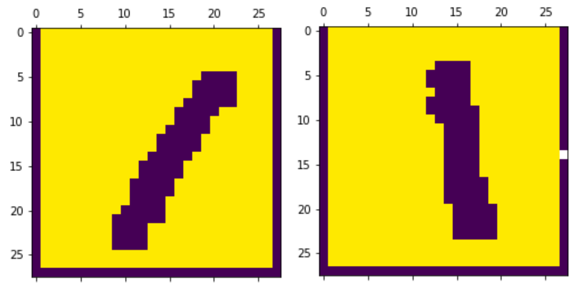
Rows of length 784 representing 28x28 images. Visualization below. (784 features)

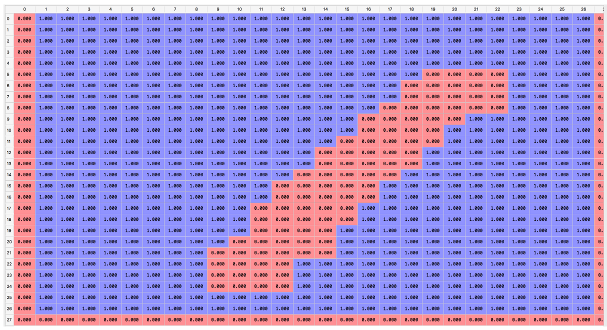
0.txt: 234 samples





1.txt: 277 samples





Visualization code snippet:

```python
def disp_img(data, n):
    y = np.empty((28, 28))
    for i in range(1, 27):
        low = 28*(i-1)
        up = 28*i -1
        for j in range(0, 27):
            y[i,j] = data[n, low:up][j]
    y = np.transpose(y)
    plt.matshow(y)
    plt.show
    return y
```

LDA – Linear Discriminant Analysis

1. Importing dataset

```python
#Importing dataset
data0 = np.genfromtxt("0.txt", delimiter = ",")
data1 = np.genfromtxt("1.txt", delimiter = ",")
```

2. Calculate mean vectors

Mean vectors for each class.

```python
#Mean vectors Class0 and Class1
m0 = np.mean(data0, 0)
m1 = np.mean(data1, 0)

mv = []
mv.append(m0)
mv.append(m1)
```

$$\mathbf{m}_j = \frac{1}{n_j} \sum_{i \in \mathcal{C}_j} x_i, \quad j = 1, 2$$

Elon Brange, 埃隆
0645120

3. Scatter matrices

a. Within-class scatter

```
#Within-class scatter matrix
#Calculating within-scatter for each class separately
#Class0
Sw0 = np.zeros((784, 784))
for i in range(1, len(data0)):
    row = data0[i,:].reshape(784,1)
    mvec = m0.reshape(784, 1)
    Sw0 += (row - mvec).dot((row - mvec).T)

#Class1
Sw1 = np.zeros((784, 784))
for i in range(1, len(data1)):
    row = data1[i,:].reshape(784,1)
    mvec = m1.reshape(784, 1)
    Sw1 += (row - mvec).dot((row - mvec).T)

#Within-scatter matrix
Sw = Sw0 + Sw1
```

Calculating within-class scatter for each class:

$$\textit{within-class scatter:}\ s_j^2 = \sum_{i \in \mathcal{C}_j} (y_i - m_j)^2, \quad j = 1, 2$$

Within-class scatter matrix for all data is: $s_1^2 + s_2^2$

→ $S_w = S_{w\ \text{class 0}} + S_{w\ \text{class 1}}$

b. Between-class scatter

```
#Between-class scatter matrix
#Calculating overall mean for each feature
mOverall = (m0 + m1)/nclass
mOverall = mOverall.reshape(784, 1)

#B/w-scatter matrix class0
mv0 = m0.reshape(784, 1)
Sb0 = (mv0 - mOverall).dot((mv0 - mOverall).T)

#B/w-scatter matrix class0
mv1 = m1.reshape(784,1)
Sb1 = (mv1 - mOverall).dot((mv1 - mOverall).T)

#Between-scatter matrix
Sb = 234*Sb0 + 277*Sb1
```

First calculate overall mean then proceeding with calculating the between-class scatter matrix:

$$S_b = \sum_{i=1}^{classes} N_i\,(\boldsymbol{m}_i - \boldsymbol{m}_{tot})(\boldsymbol{m}_i - \boldsymbol{m}_{tot})$$

Where,

N_i = sample size
$\mathbf{m}_i$ = sample mean
$\mathbf{m}_{tot}$ = overall mean

4. Solving generalized eigenvalue problem

We want to maximize between-class scatter and minimize within-class scatter:

$$J(\mathrm{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

Via Rayleigh quotient and assuming that the within-class scatter matrix (S_w) is invertible we derive the matrix:

$$\Rightarrow \mathrm{w} = S_{\mathrm{w}}^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

Equals to $S_w^{-1}S_b$

```
#Solving eigenvalue problem
#Calculating (Moore-Penrose) pseudo inverse since Sw is singluar ->
SwInv = linalg.pinv(Sw)

#Eig to sove
eigVal, eigVec = linalg.eig(SwInv.dot(Sb))
```

In this case the matrix S_w is singular and thereby not invertible and I therefore resort to calculate the pseudo inverse.

5. Selecting linear discriminants for the new subspace

```
#Eigenvalues and Eigenvectors tuples/pairs
eigPair = [(np.abs(eigVal[i]), eigVec[:,i]) for i in range(len(eigVal))]

#Sorting Eigenvalues in descending order
eigPair = sorted(eigPair, key=lambda k: k[0], reverse = True)
```

Arranging eigenvalues as tuples with their corresponding eigenvectors then sorting them in descending order. Eigenvectors with the lowest eigenvalue contains the least amount of information and should be dropped. Choosing ***k*** eigenvectors to make a 1-D transformation based on the highest eigenvalue makes us retain as much information as possible.

```
#Choosing eigenvectors with largest eigenvalues
W = np.hstack((eigPair[0][1].reshape(784,1)))
```

```
#Explanation of variance
varExpl = []
eigSum = sum(eigVal)
for i,j in enumerate(eigPair):
    varExpl.append('eigenvalue {0:}: {1:.2%}'.format(i+1, (j[0]/eigSum).real))
```

ıde ▲	Type	Size	Value
0	str	1	eigenvalue 1: 100.00%
1	str	1	eigenvalue 2: 0.00%
2	str	1	eigenvalue 3: 0.00%
3	str	1	eigenvalue 4: 0.00%
4	str	1	eigenvalue 5: 0.00%
5	str	1	eigenvalue 6: 0.00%
6	str	1	eigenvalue 7: 0.00%
7	str	1	eigenvalue 8: 0.00%

Number of linear discriminants should at most be Classes – 1 and since this is a 2-class problem it should be 1 linear discriminant (eigenvalue = 0). Resulting in only one eigenvector with eigenvalue 0 and we can form a 1-D feature space based on this eigenpair without losing a lot of information. And as can be seen by looking at the variance the first eigenpair is containing most information and a subspace based on this pair will not loose a lot of information.

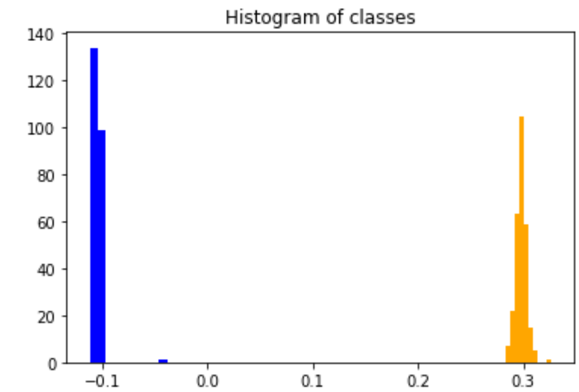
6. Transforming sample data to the new subspace

```
#Transforming data samples to new sub-space
lda0 = data0.dot(W)
lda1 = data1.dot(W)
```

Elon Brange, 埃隆
0645120

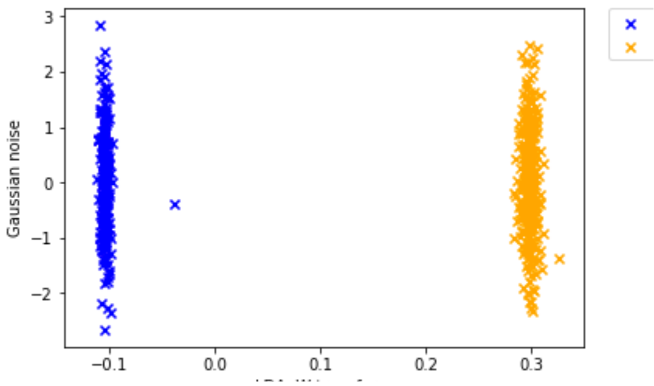
7. Visualization of result

```
#Plotting histogram of classes
plt.hist(lda0, color = "blue")
plt.hist(lda1, color = "orange")
plt.title("Histogram of classes")
plt.show()
```




```
#Plotting the samples against gaussian noise
noise0 = np.random.normal(0, 1, 234)
noise1 = np.random.normal(0, 1, 277)

plt.scatter(lda0, noise0, marker = "x", color = "blue")
plt.scatter(lda1, noise1, marker = "x", color = "orange")
plt.legend([0,1], bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel('LDA, W transform')
plt.ylabel('Gaussian noise')
plt.show()
```



Decision boundary calculated as the hyperplane between the projections of the two means.
There is no standard rule for calculating the decision boundary.

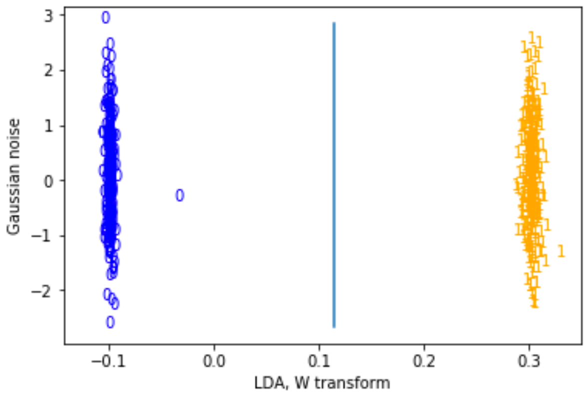
```
#Decision boundary,
bl = 0.5*(np.mean(lda1) - np.mean(lda0))

#Alternative calculation (orojecting the means)
#bl = 0.5*(m1.dot(W) - m0.dot(W))
```

```
#Alternative plot
plt.scatter(lda0, noise0, color = "white")
plt.scatter(lda1, noise1, color = "white")
for x, y in zip(lda0, noise0):
    plt.text(x, y, str(0), color="blue", fontsize=10)

for x, y in zip(lda1, noise1):
    plt.text(x, y, str(1), color="orange", fontsize=10)

plt.legend([0,1], bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.xlabel('LDA, W transform')
plt.ylabel('Gaussian noise')
plt.show()
```



Conclusions and reflections

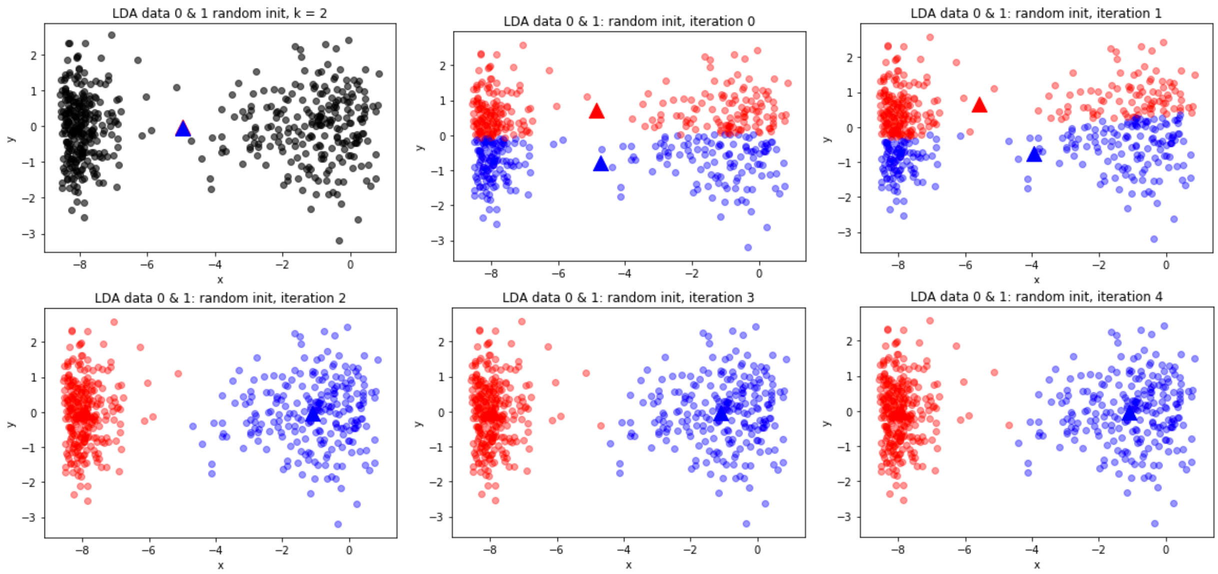
Implementing the Linear Discriminant Analysis to reduce the dimension of the provided datasets was a quite straightforward process that could, as seen above be broken down into a few steps. These steps are taken from lecture notes provided by professor Wei-Chen Chiu (Slide 180 – 184).

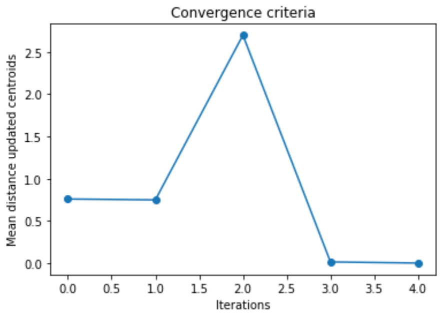
How to choose a proper decision boundary for this dataset seems to be hard. I tried several different approaches but finding a good hyper plane from the equations given above seemed impossible. With the chosen hyper plane lying between the projected means I was not satisfied and decided to see how kmeans performs on the projected dataset since it seems an easy approach and the number of clusters is already known. Further, since I choose the decision boundary to be found as a hyperplane and nothing more complex for this 2-cluster problem regular kmeans seemed to be alright. Result can be seen below.

Elon Brange, 埃隆
0645120

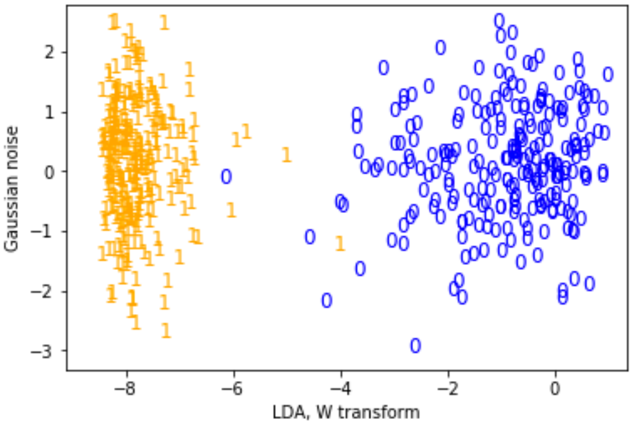
K-means clustering of the projected datasets [Not updated…]

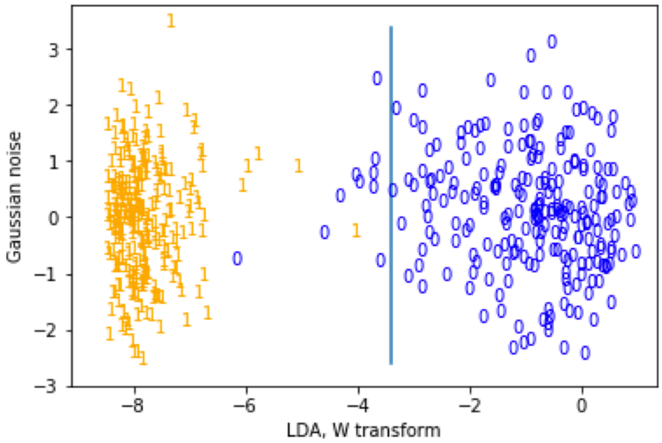
Importing the datasets and applying kmeans clustering to classify the dataset results in a correct classification rate of 0.9941291585127201%.





Elon Brange, 埃隆
0645120





Old inaccurate plots. From wrongly assuming that the matrix was symmetric.