0 – Reading data

In this section we will prepare python environment and give a function that reads the data for different types of Datasets.

```
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
```

Now we give a function that reads the data

```
def load_dataset(dataset_type = 'A', isTest = False, path = './data/hwk2data/'):
filename = 'classification' + dataset_type
if(isTest):
   filename += ".test"
else:
   filename += ".train"
data = np.loadtxt(path + filename)
X = data[:,:2]
Y = data[:,-1]
return X,Y
```

1 – Generative model (Fisher LDA)

In order to avoid any ambiguity between the number π and the parameter of the Bernoulli distribution Y, we put $\bar{\pi} = 3.1415\ldots$. We divide the indices of the training set into two classes : $\{1, 2, \ldots, n\} = \mathcal{I}_0 \cup \mathcal{I}_1$ where $y_i = 0$ for all $i \in \mathcal{I}_0$ and $y_i = 1$ for all $i \in \mathcal{I}_1$.

(a) For all $i = 1, 2, \ldots, n$ and $j = 0, 1$ we have:

$$P(X = x_i | Y = j) = \frac{1}{2\bar{\pi}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\,(x_i - \mu_j)^T \Sigma^{-1} (x_i - \mu_j)\right)$$

then log-likelihood of the parameters is given by:

$$\begin{aligned}
\ell(\pi, \mu_0, \mu_1, \Sigma) &= \log\left(p\left((x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n); \pi, \mu_0, \mu_1, \Sigma\right)\right) \\
&= \log\left(\prod_{i=1}^{n} p(x_i, y_i; \pi, \mu_0, \mu_1, \Sigma)\right) \\
&= \log\left(\prod_{i=1}^{n} p(x_i | y_i; \mu_0, \mu_1, \Sigma) p(y_i | \pi)\right) \\
&= \sum_{i=1}^{n} \log\left(p(y_i | \pi)\right) + \sum_{i=1}^{n} \log\left(p(x_i | y_i; \mu_0, \mu_1, \Sigma)\right) \\
&= \sum_{i=1}^{n} \log\left(\pi^{y_i} (1 - \pi)^{1 - y_i}\right) + \sum_{i \in \mathcal{I}_0} \log\left(\frac{1}{2\bar{\pi}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\,(x_i - \mu_0)^T \Sigma^{-1} (x_i - \mu_0)\right)\right) \\
&+ \sum_{i \in \mathcal{I}_1} \log\left(\frac{1}{2\bar{\pi}\,|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\,(x_i - \mu_1)^T \Sigma^{-1} (x_i - \mu_1)\right)\right) \\
&= \sum_{i=1}^{n} \left(y_i \log(\pi) + (1 - y_i) \log(1 - \pi)\right) - n \log(2\bar{\pi}) - \frac{n}{2} \log(|\Sigma|) \\
&- \frac{1}{2} \sum_{i \in \mathcal{I}_0} (x_i - \mu_0)^T \Sigma^{-1} (x_i - \mu_0) - \frac{1}{2} \sum_{i \in \mathcal{I}_1} (x_i - \mu_1)^T \Sigma^{-1} (x_i - \mu_1)\,.
\end{aligned}$$

The partial derivative of the log-likelihood with respect to π is given by:

$$\begin{aligned}\frac{\partial \ell(\pi,\mu_0,\mu_1,\Sigma)}{\partial \pi} &= \sum_{i=1}^{n}(\frac{y_i}{\pi}-\frac{1-y_i}{1-\pi}) \\ &= \sum_{i=1}^{n}\left(\frac{y_i}{\pi(1-\pi)}-\frac{1}{1-\pi}\right) \\ &= \frac{1}{\pi(1-\pi)}\sum_{i=1}^{n}y_i - \frac{n}{1-\pi}\end{aligned}$$

therefore the equation $\frac{\partial \ell(\pi,\mu_0,\mu_1,\Sigma)}{\partial \pi}=0$ gives $\hat{\pi}=\frac{1}{n}\sum\limits_{i=1}^{n}y_i$, which is the MLE for π. We also have,

$$\begin{aligned}\nabla_{\mu_0}\ell(\pi,\mu_0,\mu_1,\Sigma) &= -\frac{1}{2}\sum_{i\in\mathcal{I}_0}\nabla_{\mu_0}\left((x_i-\mu_0)^T\Sigma^{-1}(x_i-\mu_0)\right) \\ &= -\frac{1}{2}\sum_{i\in\mathcal{I}_0}-2\Sigma^{-1}(x_i-\mu_0) \\ &= \sum_{i\in\mathcal{I}_0}\Sigma^{-1}(x_i-\mu_0) \\ &= \Sigma^{-1}\sum_{i\in\mathcal{I}_0}x_i - |\mathcal{I}_0|\Sigma^{-1}\mu_0\end{aligned}$$

note that the equation $\nabla_{\mu_0}\ell(\pi,\mu_0,\mu_1,\Sigma)=0$ implies $\hat{\mu}_0=\frac{1}{|\mathcal{I}_0|}\sum\limits_{i\in\mathcal{I}_0}x_i$, which is the MLE of μ_0, where $|\mathcal{I}_0|$ is the cardinal of the set $\mathcal{I}_0$. Similarly the MLE of μ_1 is given by : $\hat{\mu}_1=\frac{1}{|\mathcal{I}_1|}\sum\limits_{i\in\mathcal{I}_1}x_i$. Finally let's find the MLE of Σ, note that:

$$\arg\max_{\Sigma}\ell(\pi,\mu_0,\mu_1,\Sigma)=\left(\arg\max_{\Sigma^{-1}}\ell(\pi,\mu_0,\mu_1,\Sigma)\right)^{-1}$$

therefore it is a better idea to find the MLE for Σ^{-1} since the expression of log-likelihood function depends more on Σ^{-1} which makes the computations easier, we have:

$$\begin{aligned}\nabla_{\Sigma^{-1}}\ell(\pi,\mu_0,\mu_1,\Sigma) = \frac{n}{2}\nabla_{\Sigma^{-1}}\left(\log\left(\left|\Sigma^{-1}\right|\right)\right) &- \frac{1}{2}\nabla_{\Sigma^{-1}}\left(\sum_{i\in\mathcal{I}_0}(x_i-\mu_0)^T\Sigma^{-1}(x_i-\mu_0)\right) \\ &- \frac{1}{2}\nabla_{\Sigma^{-1}}\left(\sum_{i\in\mathcal{I}_1}(x_i-\mu_1)^T\Sigma^{-1}(x_i-\mu_1)\right).\end{aligned}$$

Let $x\in\mathbb{R}^2$, we use the following well-known formulas:

$$\begin{aligned}\nabla_{\Sigma^{-1}}\log\left(\left|\Sigma^{-1}\right|\right) &= \Sigma \\ \nabla_{\Sigma^{-1}}x^T\Sigma^{-1}x &= xx^T\end{aligned}$$

therefore

$$\nabla_{\Sigma^{-1}}\ell(\pi,\mu_0,\mu_1,\Sigma)=\frac{n}{2}\Sigma-\frac{1}{2}\sum_{i=1}^{n}(x_i-\mu_{y_i})(x_i-\mu_{y_i})^T$$

the equation $\nabla_{\Sigma^{-1}}\ell(\pi,\mu_0,\mu_1,\Sigma)=0$ implies:

$$\hat{\Sigma}=\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_{y_i})(x_i-\mu_{y_i})^T$$

which is the MLE for Σ. To summarize, the MLEs of the parameters for this model are given by :

$$\begin{cases} \hat{\pi} = \frac{1}{n}\sum_{i=1}^{n} y_i \\ \hat{\mu}_0 = \frac{1}{|\mathcal{I}_0|}\sum_{i\in\mathcal{I}_0} x_i \\ \hat{\mu}_1 = \frac{1}{|\mathcal{I}_1|}\sum_{i\in\mathcal{I}_1} x_i \\ \hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_{y_i})(x_i-\mu_{y_i})^T \end{cases} \tag{1}$$

(b) Let $x = \left[x^{(1)}, x^{(2)}\right]^T \in \mathbb{R}^2$, let's calculate $p(y=1|x)$, we have :

$$\begin{aligned} p(y=1|x) &= \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)} \\ &= \frac{1}{1+\frac{1-\pi}{\pi}\frac{p(x|y=0)}{p(x|y=1)}} \\ &= \frac{1}{1+\frac{1-\pi}{\pi}\exp\left(-\frac{1}{2}\left((x-\mu_0)^T\Sigma^{-1}(x-\mu_0)-(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\right)} \\ &= \frac{1}{1+\frac{1-\pi}{\pi}\exp\left(-\frac{1}{2}\left(2(\mu_1-\mu_0)^T\Sigma^{-1}x+(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1)\right)\right)} \\ &= \frac{1}{1+\exp\left(-\left((\mu_1-\mu_0)^T\Sigma^{-1}x-\log\left(\frac{1-\pi}{\pi}\right)+\frac{1}{2}(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1)\right)\right)} \\ &= \frac{1}{1+\exp\left(-\theta^T X\right)} \end{aligned}$$

where $X = \left[1, x^{(1)}, x^{(2)}\right]^T$ and $\theta = \left[\theta^{(0)}, \theta^{(1)}, \theta^{(2)}\right]^T$ with $\theta^{(0)} = -\log\left(\frac{1-\pi}{\pi}\right)+\frac{1}{2}(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1)$ and $\left[\theta^{(1)}, \theta^{(2)}\right] = (\mu_1-\mu_0)^T\Sigma^{-1}$. The conditional distribution $p(y=1|x)$ have the form of a logistic function with parameter θ.

(c) Note that $p(y=1|x) \geq 0.5$ if and only if $\theta^T X \geq 0$. Thus, we predict $Y=1$ if $\theta^T X \geq 0$ and $Y=0$ otherwise. The line :

$$\theta^T X = 0 \Leftrightarrow x^{(2)} = -\frac{(\theta^{(0)}+\theta^{(1)}x^{(1)})}{\theta^{(2)}} \tag{2}$$

is the boundary decision that separates the data into two classes. Since we need to plot the boundary decision line, we will define two functions to do that.

♣ `line()`:

Description: The line function.

Inputs: A real number $x \in \mathbb{R}$ and the coefficients of the line $\theta \in \mathbb{R}^3$.

Outputs: A real number y in $\mathbb{R}$ according to equation (2).

```
def line(x,theta):
    y = -(theta[1]*x+theta[0])/theta[2]
return y
```

♣ `plot_decision_boundary()`:

Description: Plotting the decision boundary given model, and datasets.

Inputs: Array of datasets, array of whether to use it on test set or not, and array of models to apply.

Outputs: Plots of the datasets in $\mathbb{R}^2$ as well as the decision boundary of the models.

```
def plot_decision_boundary(dataset_types, is_test, models,
                           nrows=1, ncols=1, is_save=True, format='eps'):
  nrows = len(models)*len(is_test)
  ncols = len(dataset_types)
  counter = 0
  title = ""
  fig, axes = plt.subplots(nrows, ncols, figsize=(5.5*ncols, 4*nrows))
  if(nrows == 1 and ncols == 1):
    axes = [axes]
  else:
    axes = axes.flatten()
  for model in models:
    for test in is_test:
      for data_type in dataset_types:
        X,Y = load_dataset(data_type, is_test)
        theta = model(data_type, test)
        x_plot = np.array([np.min(X[:,0]), np.max(X[:,0])])
        y_plot = line(x_plot,theta)
        color_map = ['b' if e == 1 else 'r' for e in Y]
        axes[counter].scatter(X[:,0], X[:,1],s=2, marker = 'X',c = Y)
        axes[counter].plot(x_plot, y_plot, color='r')
        # Turn off tick labels
        axes[counter].set_yticklabels([])
        axes[counter].set_xticklabels([])
        if(test):
          title = "Test dataset " + data_type
        else:
          title = "Train dataset " + data_type
        axes[counter].set_title(title)
        counter += 1
    if(is_save):
        fig.savefig('figure_' + model.__name__ + '.' + format, format=format, bbox_inches='tight')
```

Now we implement the MLE for this generative model.

♣ `fisher_LDA()`:

Description: Implement the MLE for the generative model Fisher LDA.

Inputs: The dataset we want to classify.

Outputs: Plot the MLE of model (if show_mle=True) as in equation (1) and returns the vector θ of the line in equation (2).

```
    def fisher_LDA(dataset_type, isTest=False, show_mle=False):
        X,Y = load_dataset(dataset_type, isTest)
        X0 = X[np.where(Y == 0)]
        X1 = X[np.where(Y == 1)]
        pi = np.mean(Y)
        mu0 = np.mean(X0,axis=0)
        mu1 = np.mean(X1,axis=0)
        mu = np.append(np.tile(mu1,(X1.shape[0],1)), np.tile(mu0,(X0.shape[0],1)), axis=0)
        sigma = np.transpose(X-mu).dot(X-mu)/X.shape[0]
        if(show_mle):
           print('MLE for pi ', pi)
           print('MLE for mu0', mu0)
           print('MLE for mu1', mu1)
           print('MLE for sigma', sigma)
        theta = np.zeros(3)
        theta[1:3] = np.linalg.inv(sigma).dot(mu1-mu0)
        theta[0] = -np.log((1-pi)/pi)
                   +((np.transpose(mu0+mu1)).dot(np.linalg.inv(sigma)).dot(mu1-mu0))/2
        return (theta)
```

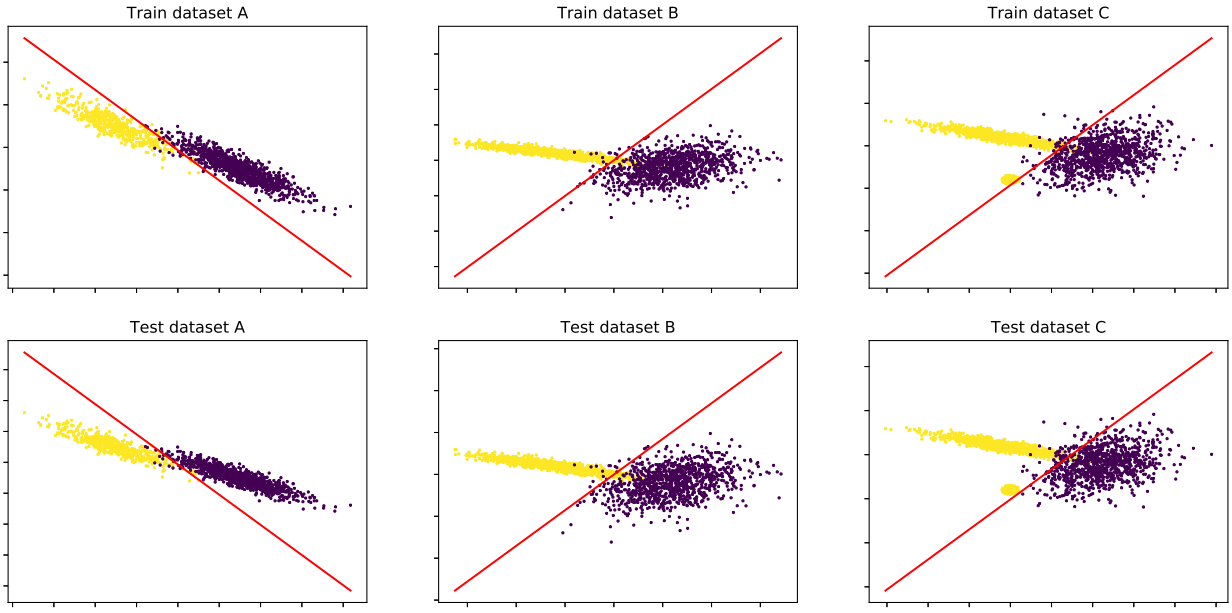

Figure 1: Decision boundary of Fisher LDA model on the three datasets

The numerical values of the MLEs for each training dataset are given by :

$$\begin{cases} \hat{\mu}^{(\mathrm{A})} \approx 0.33 \\ \hat{\mu}_0^{(\mathrm{A})} \approx [2.89, -0.89]^T \\ \hat{\mu}_1^{(\mathrm{A})} \approx [-2.69, 0.86]^T \\ \hat{\Sigma}^{(\mathrm{A})} \approx \begin{pmatrix} 2.44 & -1.13 \\ -1.13 & 0.61 \end{pmatrix} \end{cases}, \quad \begin{cases} \hat{\mu}^{(\mathrm{B})} \approx 0.5 \\ \hat{\mu}_0^{(\mathrm{B})} \approx [3.34, -0.83]^T \\ \hat{\mu}_1^{(\mathrm{B})} \approx [-3.21, 1.08]^T \\ \hat{\Sigma}^{(\mathrm{B})} \approx \begin{pmatrix} 3.34 & -0.13 \\ -0.13 & 1.73 \end{pmatrix} \end{cases} \text{ and } \begin{cases} \hat{\mu}^{(\mathrm{C})} \approx 0.62 \\ \hat{\mu}_0^{(\mathrm{C})} \approx [2.79, -0.83]^T \\ \hat{\mu}_1^{(\mathrm{C})} \approx [-2.94, -0.95]^T \\ \hat{\Sigma}^{(\mathrm{C})} \approx \begin{pmatrix} 17.88 & 4.12 \\ 4.12 & 5.39 \end{pmatrix} \end{cases}$$

The graphical representation of the all datasets with the decision boundary is shown in Figure 1.

2 – Logistic regression

Put $\theta = (b, w^{(1)}, \dots, w^{(d)})^T \in \mathbb{R}^{d+1}$ and $x = (1, x^{(1)}, \dots, x^{(d)})^T \in \mathbb{R}^{d+1}$. Let

$$g_\theta(x) = p(y = 1|x) = \frac{1}{1 + \exp\left(-f_\theta(x)\right)}$$

be the prediction function, where $f_\theta(x) = w^T x + b = \theta^T x$. Then the log-likelihood function is defined by :

$$\begin{aligned} \ell(\theta) &= \log\left(\prod_{i=1}^{n} p(y_i|x_i)\right) \\ &= \log\left(\prod_{i=1}^{n} g_\theta(x_i)^{y_i}(1 - g_\theta(x_i))^{1-y_i}\right) \\ &= \sum_{i=1}^{n} y_i \log\left(g_\theta(x_i)\right) + (1 - y_i)\log(1 - g_\theta(x_i)) \\ &= \sum_{i=1}^{n} \left(-y_i \log(1 + e^{-\theta^T x_i}) + (1 - y_i)\log\left(\frac{e^{-\theta^T x_i}}{1 + e^{-\theta^T x_i}}\right)\right) \\ &= \sum_{i=1}^{n} \left(y_i \theta^T x_i - \log\left(1 + e^{\theta^T x_i}\right)\right) \end{aligned}$$

Thus the gradient with respect to θ of $\ell(\theta)$ is given by:

$$\begin{aligned}\nabla\ell(\theta) &= \sum_{i=1}^{n}\left(y_i x_i - \frac{e^{\theta^T x_i}}{1+e^{\theta^T x_i}}x_i\right)\\ &= \sum_{i=1}^{n}\left(y_i - g_\theta(x)\right)x_i\end{aligned}$$

Furthermore, the hessian matrix is given by

$$H(\ell(\theta)) = -\sum_{i=1}^{n} x_i x_i^T g_\theta(x_i)\left(1-g_\theta(x_i)\right)$$

As seen in class, define $X = [x_1,\dots,x_n]^T \in \mathbb{R}^{n\times(d+1)}$, $\mathbf{y} = [y_1,\dots,y_n]^T \in \mathbb{R}^n$ and $\boldsymbol{\mu} = [\mu_1,\dots,\mu_n]^T \in \mathbb{R}^n$ where $\mu_i = g_\theta(x_i)$. Then we can rewrite the gradient and the hessian of $\ell(\theta)$ as :

$$\nabla\ell(\theta) = X^T(y-\mu) \text{ and } H(\ell(\theta)) = -X^T D X$$

where D is a diagonal matrix defined by $D_{ii} = \mu_i(1-\mu_i)$. Newton's method for maximizing the log-likelihood of $\ell(\theta)$ is given by :

$$\begin{cases}\theta^{(0)} = \vec{0}\\ \theta^{(k+1)} = \theta^{(k)} - \left(H(\ell(\theta^{(k)}))\right)^{-1}\nabla\ell(\theta^{(k)})\end{cases}$$

♣ `logistic_regression()`:

Description: Implement the logistic regression model.

Inputs: The dataset we want to classify.

Outputs: The vector θ of the line in equation (2).

```
def logistic_regression(dataset_type, isTest=False):
    X,Y = load_dataset(dataset_type, isTest)
    error = 1e-10
    maxit = 10000
    X = np.concatenate((np.ones((X.shape[0],1)),X),axis=1)
    theta = np.zeros(X.shape[1])
    for i in range(maxit):
        mu = 1/(1 + np.exp(-X.dot(theta)))
        D = np.diag(mu*(1-mu))
        Grad = np.transpose(X).dot(Y-mu)
        Hess = -np.transpose(X).dot(D).dot(X)
        ew_theta = theta - np.linalg.lstsq(Hess,Grad,rcond=-1)[0]
        if( np.linalg.norm(new_theta - theta) < error ):
            return (new_theta)
        theta = new_theta
    return (theta)
```

(a) The numerical values of the learnt parameters are

$$\begin{cases}b^{(A)} \approx -259.43, & w^{(A)} \approx [-1475.35, -2557.22]^T\\ b^{(B)} \approx 1.34, & w^{(B)} \approx [-1.70, 1.02]^T\\ b^{(C)} \approx 0.96, & w^{(C)} \approx [-2.20, 0.71]^T\end{cases}$$

(b) Note that similarly to the question 1 $p(y=1|x) > 0.5$ if and only if $\theta^T X > 0$. Thus, we use the function `plot_decision_boundary()` for each dataset type. Figure 2 shows the graphical representation of the all datasets with the decision boundary.

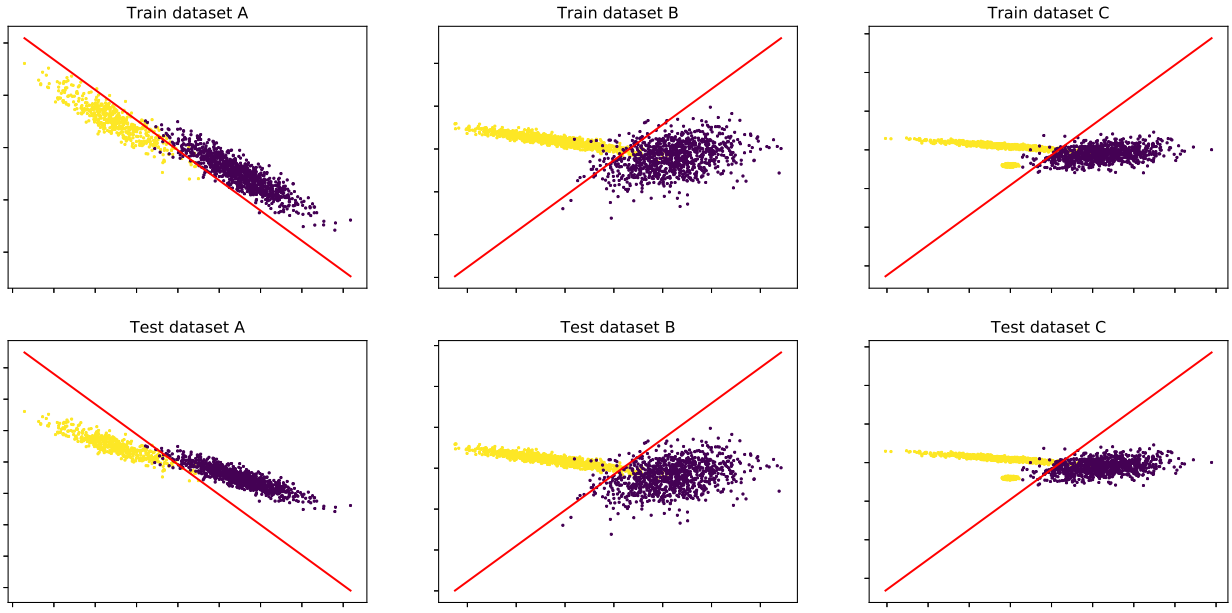

Figure 2: Decision boundary of Logistic Regression model on the three datasets

3 – Linear regression

In this question we will use the same notations as in question 2, but here the prediction function is given by $f_\theta(x) = \theta^T x$. In class we've seen that the optimal parameter θ is given by:

$$\begin{aligned}\hat{\theta} &= \arg\min_\theta \sum_{i=1}^{n} (y_i - \theta^T x_i) \\ &= (X^T X)^{-1} X^T \mathbf{y}\end{aligned}$$

Note that in this model, the boundary decision line is given by $f_\theta(x) = 0.5$ which is equivalent to

$$(\theta^{(0)} - 0.5) + \theta^{(1)} x^{(1)} + \theta^{(2)} x^{(2)} = 0 \tag{3}$$

♣ `Logistic_regression()`:

Description: Implement the linear regression model.

Inputs: The dataset we want to classify.

Outputs: The vector $[\theta^{(0)} - 0.5, \theta^{(1)}, \theta^{(2)}]$ of the line in equation (3).

```
def linear_regression(dataset_type, isTest=False):
    X,Y = load_dataset(dataset_type, isTest)
    X = np.concatenate((np.ones((X.shape[0],1)),X),axis=1)
    theta=(np.linalg.inv(np.transpose(X).dot(X))).dot(np.transpose(X)).dot(Y)
    theta[0]=theta[0]-0.5
    return (theta)
```

(a) The numerical values of the learnt parameters are

$$\begin{cases} b^{(A)} \approx 0.49, & w^{(A)} \approx [-0.26, -0.37]^T \\ b^{(B)} \approx 0.50, & w^{(B)} \approx [-0.10, 0.05]^T \\ b^{(C)} \approx 0.51, & w^{(C)} \approx [-0.13, -0.02]^T \end{cases}$$

(b) By using the function `plot_decision_boundary()` we get the graphs on Figure 3 that shows the graphical representation of the all datasets with the decision boundary.

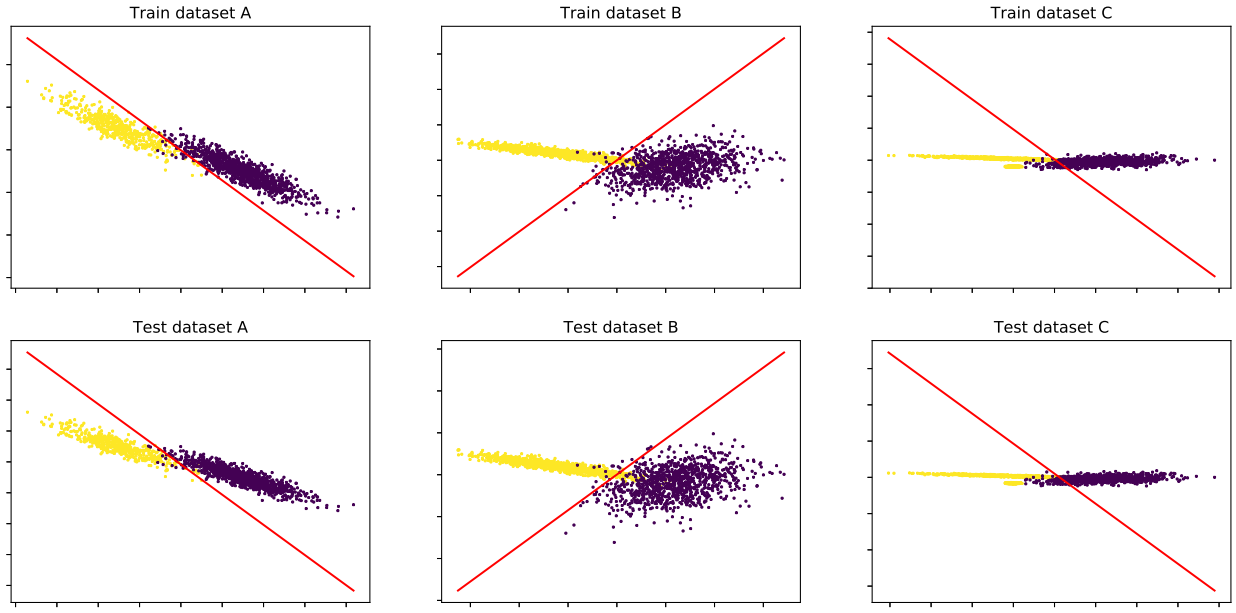

Figure 3: Decision boundary of Linear Regression model on the three datasets

4 – Misclassification error

Let n be the length of the dataset, denote by Y^* the vector of predicted labels using a certain model and Y the true labels on the dataset, consider $m = \{i = 1, 2, \ldots, n \mid Y_i^* = Y_i\}$ then the misclassification error is given by:

$$\epsilon = 1 - \frac{m}{n} \tag{4}$$

We use the following function to compute the misclassification error:
♣ `misclassification():`

Description: Calculate the misclassification error of a certain method.

Inputs: The dataset we are interested in as well as the used method.

Outputs: The misclassification error according to equation (4).

```
def misclassification(dataset_type, isTest, function):
  X,Y = load_dataset(dataset_type, isTest)
  X = np.concatenate((np.ones((X.shape[0],1)), X), axis=1)
  theta = function(dataset_type, isTest=False)
  prediction = list(map(int, (X.dot(theta))>=0))
  missc = 1-np.sum(Y == prediction)/X.shape[0]
  return(missc)
```

(a) By using the function `misclassification()` for each dataset, we get the following table :

	Fisher LDA		**Logistic regression**		**Linear regression**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Dataset A	0.67%	2.07%	0.00%	3.53%	1.33%	2.07%
Dataset B	3.00%	4.15%	2.00%	4.30%	3.00%	4.15%
Dataset C	3.75%	2.70%	4.00%	2.27%	5.50%	4.23%

Table 1: Misclassification error table for different dataset types.

(b) Comparaison between the different models:

- **Comparaison between the Logistic and Linear regresion**:
  For datasets A and B, Logistic regression makes less error on the training set that Linear regression, while making more error on the test dataset. Which means that for those two models, linear regression is the one that succeeded to generalized better.
  For dataset C, Logistic Regression makes less error on both the training set, and significantly less error on the test set, even when compared to all other models in the assignment.

- **Comparaison between the LDA v.s Logistic and Linear regression**:
  For dataset A, LDA makes the assumptions verified for this dataset, so its test error is lowest, even though Linear regression does just as well as LDA on test set but worst on the training set. The Logistic regression does very well on the training set with 0% error, but very bad comapred to the other two methodes (3.5% error).
  For dataset B, LDA does just as well as Linear Reg, and again, Logistic Reg. has smallest training error, but largest test error.
  On Dataset C, LDA and Logistic regression have close results were not bad, compared to linear regression, with largest misclassification error on both training and test set.

This means that the choice of the dataset is what determines the performance of a classifier, as LDA makes the assumption that the data is drawn from a Gaussian with different means but same covariance matrix, while the other two regression models just try to seperate data using a line with no further assumptions, this turns out to work because in the graphs we can see that for all previous datasets, we can visually find a line that seperates the two classes within them with low misclassification error.

5 – QDA model

(a) We will use the same notations as in question 1. For all $i = 1, 2, \ldots, n$ and $j = 0, 1$ we have:

$$P(X = x_i | Y = j) = \frac{1}{2\bar{\pi} \left|\Sigma_j\right|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\left(x_i - \mu_j\right)^T \Sigma_j^{-1} (x_i - \mu_j)\right)$$

then similarly to question 1, the log-likelihood function can be expressed as:

$$\begin{aligned}\ell(\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1) = &\sum_{i=1}^{n} \left(y_i \log(\pi) + (1 - y_i) \log(1 - \pi)\right) - n \log(2\bar{\pi}) - \frac{|\mathcal{I}_0|}{2} \log(|\Sigma_0|) - \frac{|\mathcal{I}_1|}{2} \log(|\Sigma_1|) \\ &- \frac{1}{2} \sum_{i \in \mathcal{I}_0} (x_i - \mu_0)^T \Sigma_0^{-1} (x_i - \mu_0) - \frac{1}{2} \sum_{i \in \mathcal{I}_1} \left(x_i - \mu_1\right)^T \Sigma_1^{-1} (x_i - \mu_1) \,.\end{aligned}$$

by using the same arguments as in question 1, we can conclude that the MLEs for the parameters of this model are given by:

$$\begin{cases} \hat{\pi} = \frac{1}{n} \sum_{i=1}^{n} y_i \\ \hat{\mu}_0 = \frac{1}{|\mathcal{I}_0|} \sum_{i \in \mathcal{I}_0} x_i \\ \hat{\mu}_1 = \frac{1}{|\mathcal{I}_1|} \sum_{i \in \mathcal{I}_1} x_i \\ \hat{\Sigma}_0 = \frac{1}{|\mathcal{I}_0|} \sum_{i \in \mathcal{I}_0} (x_i - \mu_0)(x_i - \mu_0)^T \\ \hat{\Sigma}_1 = \frac{1}{|\mathcal{I}_1|} \sum_{i \in \mathcal{I}_1} (x_i - \mu_1)(x_i - \mu_1)^T \end{cases} \tag{5}$$

(b) Graphical representation of the decision boundary. We start by calculating $p(y=1|x)$:

$$\begin{aligned} p(y=1|x) &= \frac{1}{1+\frac{1-\pi}{\pi}\frac{p(x|y=0)}{p(x|y=1)}} \\ &= \frac{1}{1+\frac{1-\pi}{\pi}\left|\frac{\Sigma_1}{\Sigma_0}\right|^{\frac{1}{2}}\exp\left(-\frac{1}{2}\left((x-\mu_0)^T\Sigma_0^{-1}(x-\mu_1)-(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right)\right)} \\ &= \frac{1}{1+\exp\left(-\frac{1}{2}\left((x-\mu_0)^T\Sigma_0^{-1}(x-\mu_1)-(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right)+\log\left(\frac{1-\pi}{\pi}\right)+\frac{1}{2}\log\left(\left|\frac{\Sigma_1}{\Sigma_0}\right|\right)\right)} \\ &= \frac{1}{1+\exp\left(-\left(x^TAx+w^Tx+b\right)\right)} \end{aligned}$$

where :

$$\begin{cases} A = \Sigma_0^{-1}-\Sigma_1^{-1} \\ w = -2\left(\Sigma_0^{-1}\mu_0-\Sigma_1^{-1}\mu_1\right) \\ b = \mu_0^T\Sigma_0^{-1}\mu_0-\mu_1^T\Sigma_1^{-1}\mu_1+\log|\Sigma_0|-\log|\Sigma_1|+2\log\left(\frac{1-\pi}{\pi}\right) \end{cases}.$$

Moreover, $p(y=1|x)=0.5$ if and only if $x^TAx+w^Tx+b=0$, thus the boundary decision curve have the form of a conic.

Using the function `qda_model` shown below, the learnt parameters are computed, and also given below.
♣ `qda_model()`:

Description: Implement the QDA model.

Inputs: The dataset we want to classify.

Outputs: The tuple $(\pi, \mu_0, \mu_1, \Sigma_0, \Sigma_1)$ of estimated parameters in (5).

```
def qda_model(dataset_type, isTest=False):
  X,Y = load_dataset(dataset_type, isTest)
  X0 = X[np.where(Y == 0)]
  X1 = X[np.where(Y == 1)]
  pi = np.mean(Y)
  mu0 = np.mean(X0,axis=0)
  mu1 = np.mean(X1,axis=0)
  sigma0 = np.transpose(X0-mu0).dot(X0-mu0)/X0.shape[0]
  sigma1 = np.transpose(X1-mu1).dot(X1-mu1)/X1.shape[0]
  return (pi,mu0,mu1,sigma0,sigma1)
```

(a) The numerical values of the learnt parameters are

$$\begin{cases} \hat{\mu}^{(A)} \approx 0.33 \\ \hat{\mu}_0^{(A)} \approx [2.9, -0.89]^T \\ \hat{\mu}_1^{(A)} \approx [-2.69, 0.87]^T \\ \hat{\Sigma}_0^{(A)} \approx \begin{pmatrix} 2.31 & -1.05 \\ -1.05 & 0.58 \end{pmatrix} \\ \hat{\Sigma}_1^{(A)} \approx \begin{pmatrix} 2.7 & -1.3 \\ -1.3 & 0.69 \end{pmatrix} \end{cases} \quad \begin{cases} \hat{\mu}^{(B)} \approx 0.5 \\ \hat{\mu}_0^{(B)} \approx [3.34, -0.84]^T \\ \hat{\mu}_1^{(B)} \approx [-3.22, 1.08]^T \\ \hat{\Sigma}^{(B)} \approx \begin{pmatrix} 2.54 & 1.06 \\ 1.06 & 2.96 \end{pmatrix} \\ \hat{\Sigma}_1^{(B)} \approx \begin{pmatrix} 4.15 & -1.33 \\ -1.33 & 0.52 \end{pmatrix} \end{cases} \quad \text{and} \quad \begin{cases} \hat{\mu}^{(C)} \approx 0.62 \\ \hat{\mu}_0^{(C)} \approx [2.79, -0.84]^T \\ \hat{\mu}_1^{(C)} \approx [-2.94, -0.96]^T \\ \hat{\Sigma}^{(C)} \approx \begin{pmatrix} 2.9 & 1.25 \\ 1.25 & 2.92 \end{pmatrix} \\ \hat{\Sigma}_1^{(C)} \approx \begin{pmatrix} 2.87 & -1.76 \\ -1.76 & 6.56 \end{pmatrix} \end{cases}$$

(b) By using the function `plot_decision_boundary_qda()`, we get the graphs on Figure 4 that shows the graphical representation of the all datasets with the decision boundary.

♣ `plot_decision_boundary_qda()`:

Description: Plot the datasets along with the decision boundary of QDA model.

Inputs: The datasets we want to classify.

Outputs: The plots of datasets with decision boundary.

```
def plot_decision_boundary_qda(dataset_types, is_test, model,
                        nrows=1, ncols=1, is_save=False, format='eps'):
nrows = len(is_test)
ncols = len(dataset_types)
counter = 0
title = ""
fig, axes = plt.subplots(nrows, ncols, figsize=(5.5*ncols, 4*nrows))
if(nrows == 1 and ncols == 1):
    axes = [axes]
else:
    axes = axes.flatten()
for test in is_test:
    for data_type in dataset_types:
        X,Y = load_dataset(data_type, test)
        (pi,mu0,mu1,sigma0,sigma1) = qda_model(data_type, test)
        A = np.linalg.inv(sigma0)-np.linalg.inv(sigma1)
        a = -2*(np.linalg.inv(sigma0).dot(mu0)-np.linalg.inv(sigma1).dot(mu1))
        alpha = np.transpose(mu0).dot(np.linalg.inv(sigma0)).dot(mu0)\
        -np.transpose(mu1).dot(np.linalg.inv(sigma1)).dot(mu1)\
        +2*np.log((1-pi)/pi)+np.log(np.linalg.det(sigma0))\
        -np.log(np.linalg.det(sigma1))
        x_plot = np.linspace(np.min(X[:,0]), np.max(X[:,0]),100).reshape([-1,1])
        y_plot = np.linspace(np.min(X[:,1]), np.max(X[:,1]),100).reshape([-1,1])
        x_plot, y_plot = np.meshgrid(x_plot, y_plot)
        Z = np.zeros(x_plot.shape)
        for i in range(x_plot.shape[0]):
            for j in range(y_plot.shape[1]):
                vec = np.asarray([x_plot[i,j], y_plot[i,j]]).reshape([2,1])
                Z[i,j] = vec.transpose().dot(A).dot(vec)+a.transpose().dot(vec)+alpha
        axes[counter].contour(x_plot, y_plot, (Z), [0], colors='r')
        axes[counter].scatter(X[:,0], X[:,1], s=2, marker = 'X', c=Y)
        # Turn off tick labels
        axes[counter].set_yticklabels([])
        axes[counter].set_xticklabels([])
        if(test):
            title = "Test dataset " + data_type
        else:
            title = "Train dataset " + data_type
        axes[counter].set_title(title)
        counter += 1
if(is_save):
    fig.savefig('figure_' + model.__name__ + '.' + format, format=format, bbox_inches='tight')
```

(c) Table 2 shows the misclassification error that is calculated using the function `misclassification_qda()`. Its code is given below.

♣ `plot_decision_boundary_qda()`:

Description: Calculates the misclassification error on a given dataset for QDA model.

Inputs: The dataset on which we want to get the misclassification error of QDA.

Outputs: The misclassification error.

```
def misclassification_qda(dataset_type, isTest):
  X,Y = load_dataset(dataset_type, isTest)
  (pi,mu0,mu1,sigma0,sigma1) = qda_model(dataset_type,isTest=False)
  A = np.linalg.inv(sigma0)-np.linalg.inv(sigma1)
  a = -2*(np.linalg.inv(sigma0).dot(mu0)-np.linalg.inv(sigma1).dot(mu1))
  alpha = np.transpose(mu0).dot(np.linalg.inv(sigma0)).dot(mu0)\
  -np.transpose(mu1).dot(np.linalg.inv(sigma1)).dot(mu1)+2*np.log((1-pi)/pi)\
  +np.log(np.linalg.det(sigma0))-np.log(np.linalg.det(sigma1))
  prediction = np.zeros(Y.shape)
```

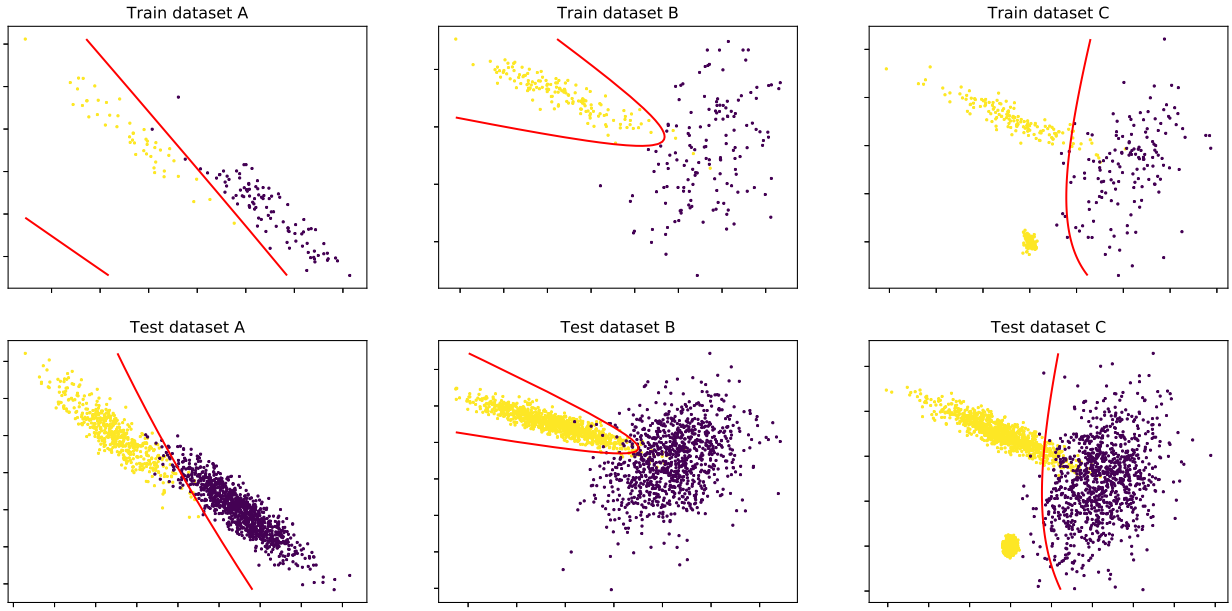

Figure 4: Decision boundary of QDA model applied on the three datasets

	QDA model	
	Train	**Test**
Dataset A	1.33%	2.73%
Dataset B	1.33%	2.00%
Dataset C	5.00%	3.60%

Table 2: Misclassification error table for different dataset types.

```
    for i in range (X.shape[0]):
      prediction[i] = int(X[i,:].transpose().dot(A).dot(X[i,:])\
      +a.transpose().dot(X[i,:])+alpha>=0)
    missc = 1-np.sum(Y == prediction)/X.shape[0]
    return(missc)
```

(c) We expect to see that QDA outperforms all the three previous methods, but that's not always the case

(d) As expected, the QDA does the smallest error (both on training and test sets) on dataset B, because the assumptions of the source of this dataset match QDA's. On the other hand, it didn't do astoundingly well on the other datasets compared with the rest of the models. The other models outperforms it on dataset A. On dataset C, It does worst that LDA and Logistic regression even though QDA has higher capacity, but performs better that linear regression.