Question 4: Gaussian Discriminant Analysis

Data

In the current problem we are given m values of attributes such that for each i^{th} value of the attribute ($\alpha^i \epsilon R^2$) we have $Y^i \epsilon R^1$. We define X^i such that for each i^{th} sample X^i $=< \alpha^i >$ where $X \epsilon R^2$.

Normal Equations used to calculate model parameters: θ <Σ Σ_1 Σ_2 μ_1 μ_2 ϕ>

Model : $P(Y = k|X;\theta) = \dfrac{\phi^k(1-\phi)^{1-k}\frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}}e^{-\frac{(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k)}{2}}}{\sum_{i=0}^{1}(\phi^i(1-\phi)^{1-i}\frac{1}{(2\pi)^{n/2}|\Sigma_i|^{1/2}}e^{-\frac{(X-\mu_i)^T\Sigma_i^{-1}(X-\mu_i)}{2}})}$

Estimate for ϕ **:** $\phi = \dfrac{\sum_{i=1}^{m}\mathbb{1}\{Y^i=1\}}{\sum_{i=1}^{m}\mathbb{1}}$

Estimate for μ_k**:** $\mu_k = \dfrac{\sum_{i=1}^{m}\mathbb{1}\{Y^i=K\}X^i}{\sum_{i=1}^{m}\mathbb{1}\{Y^i=K\}}$

Estimate for Σ_k**:** $\Sigma_k = \dfrac{\sum_{i=1}^{m}\mathbb{1}\{Y^i=K\}(X-\mu_k)^T(X-\mu_k)}{\sum_{i=1}^{m}\mathbb{1}\{Y^i=K\}}$

Estimate for Σ**:** $\Sigma = \dfrac{\sum_{K=0}^{1}\sum_{i=1}^{m}\mathbb{1}\{Y^i=K\}(X^i-\mu_k)^T(X^i-\mu_k)}{\sum_{1}^{m}\mathbb{1}}$

In [69]:

```
import sys
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from time import time
import warnings
warnings.filterwarnings("ignore")
#%matplotlib
```

executed in 13ms, finished 20:55:25 2018-02-12

Loading training and testing data

In [70]:

```
X = pd.read_csv('q4x.dat', delimiter=' ').values[:,0::2]
Y = pd.read_csv('q4y.dat', delimiter="\t").values
_1 = np.ones(Y.shape)
X0 = X[Y[:,0] == 'Canada']
X1 = X[Y[:,0] == 'Alaska']
```

executed in 317ms, finished 20:55:25 2018-02-12

Calculating model estimates for case when $\Sigma_0 = \Sigma_1 = \Sigma$

In [71]:

```
phi = np.sum(_1[Y[:,0] == 'Alaska']) / np.sum(_1)
mu_1 = np.sum(X1,axis=0) / np.sum(_1[Y[:,0] == 'Alaska']).reshape(1,-1)
mu_0 = np.sum(X0,axis=0) / np.sum(_1[Y[:,0] == 'Canada']).reshape(1,-1)

Cx = (np.dot((X1-mu_1).T,(X1-mu_1))+np.dot((X0-mu_0).T,(X0-mu_0)))/np.sum(_1)

# print("%s = "%(u'\u03A6'))
# print(phi)
# print("%s = "%( u'\u03A3'))
# print(Cx)
# print("%s1 = "%(u'\u03BC'))
# print(mu_1)
# print("%s0 = "%(u'\u03BC'))
# print(mu_0)
```

executed in 263ms, finished 20:55:26 2018-02-12

Model estimates

Φ =**0.494949494949495**

Σ =**[[289.43198928 -20.90429602] [-20.90429602 1095.4086209]]**

μ_1 =**[[98.18367347 430.91836735]]**

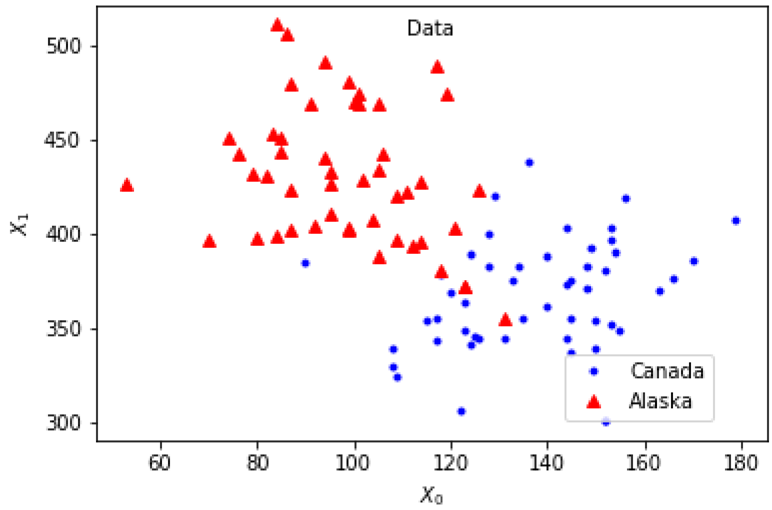
μ_0 =**[[137.46 366.62]]**

Modeling data

In [72]:

```
plt.figure(1)
plt.title("Data",fontsize=10,y=0.9)
X0 = X[Y[:,0] == 'Canada']
X1 = X[Y[:,0] == 'Alaska']

plt.plot(X0[:,0],X0[:,1],'b.',label='Canada')
plt.plot(X1[:,0],X1[:,1],'r^',label='Alaska')
plt.legend(bbox_to_anchor=(0.7, 0.2), loc=2, borderaxespad=0.)
plt.xlabel(r'$X_{0}$')
plt.ylabel(r'$X_{1}$')
plt.draw()
```

executed in 495ms, finished 20:55:26 2018-02-12



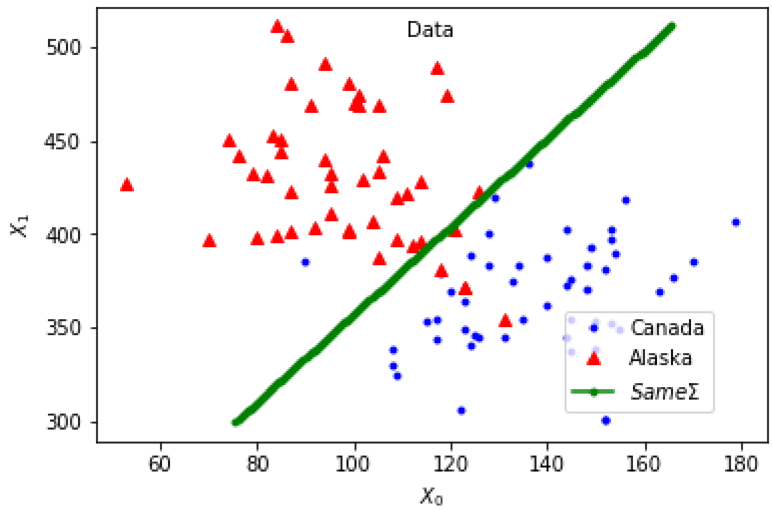
Plotting bondary separating two classes with same Σ

Separating boundary is defined by $P(Y = 1|X;\theta) = P(Y = 0|X;\theta)$
Separation boundary is linear i.e. $2 * X\Sigma^{-1}(\mu_1 - \mu_0)^T + C = 0$ where
$C = (\mu_1 - \mu_0)\Sigma^{-1}(\mu_1 - \mu_0)^T - 2log(\frac{\phi}{1-\phi})$

In [73]:

```
Cxinv = np.linalg.pinv(Cx)
C = np.dot(mu_1, np.dot(Cxinv, mu_1.T)) - np.dot(mu_0, np.dot(Cxinv, mu_0.T)) - 2*n
M = 2*np.dot(Cxinv,(mu_1-mu_0).T)
M_same = M.copy()
C_same = C.copy()

plt.figure(1)
plt.title("Data",fontsize=10,y=0.9)

plt.plot(X0[:,0],X0[:,1],'b.',label='Canada')
plt.plot(X1[:,0],X1[:,1],'r^',label='Alaska')

minX = int(np.min(X[:,1]))-1
maxX = int(np.max(X[:,1]))+1

Xp = np.asarray([x for x in range(minX,maxX)])

Xo = ((-1*(Xp*M[1,0])+C)/(M[0,0]))
plt.plot(Xo[0],Xp,'.g-',label=r'$Same \Sigma$')
plt.legend(bbox_to_anchor=(0.7, 0.3), loc=2, borderaxespad=0.)
plt.xlabel(r'$X_{0}$')
plt.ylabel(r'$X_{1}$')
plt.draw()
```

executed in 432ms, finished 20:55:27 2018-02-12



Calculating model estimates for case when $\Sigma_0 \neq \Sigma_1$

In [74]:

```
Cx1 = np.dot((X1-mu_1).T,(X1-mu_1))/np.sum(_1[Y[:,0] == 'Alaska'])
Cx0 = np.dot((X0-mu_0).T,(X0-mu_0))/np.sum(_1[Y[:,0] == 'Canada'])

# print("%s = "%(u'\u03A6'))
# print(phi)
# print("%s1 = "%( u'\u03A3'))
# print(Cx1)
# print("%s1 = "%(u'\u03BC'))
# print(mu_1)

# print("%s0 = "%( u'\u03A3'))
# print(Cx0)
# print("%s0 = "%(u'\u03BC'))
# print(mu_0)
```

executed in 51ms, finished 20:55:27 2018-02-12

Model estimates

Φ **=0.494949494949495**

Σ_1 **=[[258.68054977 -175.74010829] [-175.74010829 1319.91170346]]**

μ_1 **=[[98.18367347 430.91836735]]**

Σ_0 **=[[319.5684 130.8348] [130.8348 875.3956]]**

μ_0 **=[[137.46 366.62]]**

Calculating parameters of boundary equation

Separating boundary is defined by $P(Y=1|X;\theta) = P(Y=0|X;\theta)$
Separation boundary is Quadratic i.e. $X(\Sigma_1^{-1} - \Sigma_0^{-1})X^T - 2X(\Sigma_1^{-1}(\mu_1^T) - \Sigma_0^{-1}(\mu_0^T)) + C = 0 \quad (1)$
where $C = (\mu_1)\Sigma_1^{-1}(\mu_1)^T - (\mu_0)\Sigma_0^{-1}(\mu_0)^T + log(\frac{|\Sigma_1|}{|\Sigma_0|}) - 2log(\frac{\phi}{1-\phi})$

Approach 1 Using normal equation

Now, $X =< X_0, X_1 >$ for decision boundary we can define (1) as $XAX^T - XB + C = 0 \quad (2)$ where,
$A = \Sigma_1^{-1} - \Sigma_0^{-1}$
$B = -2(\Sigma_1^{-1}(\mu_1^T) - \Sigma_0^{-1}(\mu_0^T))$
$C = C$
We can write equation (2) as
$X_0A_{00}X_0 + X_1A_{10}X_0 + X_0A_{01}X_1 + X_1A_{11}X_1 + X_0B_0 + X_1B_1 + C = 0$
or
$X_0^2A_{00} + X_0(2X_1A_{10} + B_0) + (X_1A_{11}X_1 + X_1B_1 + C) = 0$
For a given X_1 equation 2 will be a quadratic in terms of X_0 so solve the equation to get X_0 all such pair of $< X_0, X_1 >$ defines our boundary

In [ ]:

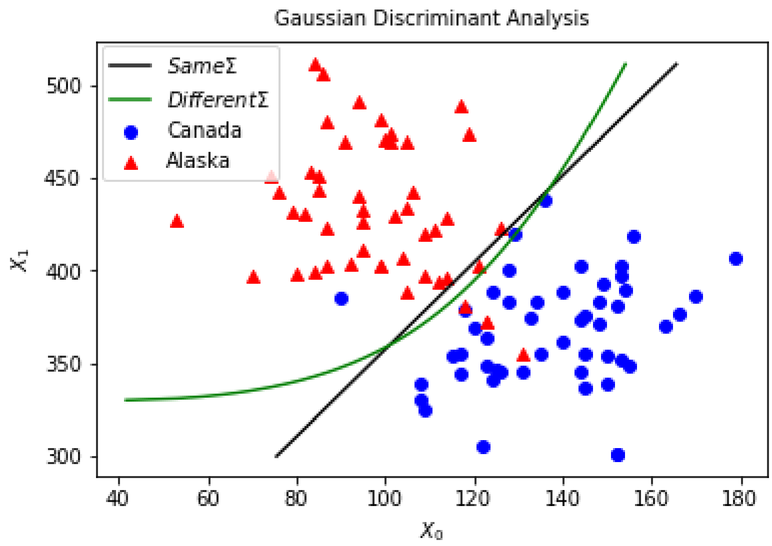
```
Cxinv1 = np.linalg.pinv(Cx1)
Cxinv0 = np.linalg.pinv(Cx0)
C = np.dot(mu_1, np.dot(Cxinv1, mu_1.T)) - np.dot(mu_0, np.dot(Cxinv0, mu_0.T))+np.
M = -2*(np.dot(Cxinv1,mu_1.T)-np.dot(Cxinv0,mu_0.T))

A = Cxinv1-Cxinv0

def find_X0(X1,A,M,C):
    c = C + (X1*X1)*A[1,1] + X1*M[1,0]
    b = X1*A[0,1] + X1*A[1,0] + M[0,0]
    a = A[0,0]
    X2 = (-1*b + np.sqrt((b**2)-(4*a*c)))/(2*a)
#     Taking only larger value
    return X2
```

Plotting data and learned model for cases $\Sigma_0 = \Sigma_1$ (Same Σ) and $\Sigma_0 \neq \Sigma_1$ (Different Σ)

In [115]:

```
plt.figure(4)
plt.cla()
X0 = X[Y[:,0] == 'Canada']
X1 = X[Y[:,0] == 'Alaska']
plt.title("Gaussian Discriminant Analysis",fontsize=10,y=1.008)
plt.scatter(X0[:,0],X0[:,1],c='b',marker='o',label='Canada')
plt.scatter(X1[:,0],X1[:,1],c='r',marker='^',label='Alaska')

minX = int(np.min(X[:,1]))-1
maxX = int(np.max(X[:,1]))+1


Xo = ((-1*(Xp*M_same[1,0])+C_same)/(M_same[0,0]))
plt.plot(Xo[0],Xp,'k-',label=r'$Same \Sigma$')

Xp = np.asarray([x for x in range(minX,maxX)])
Xo = find_X0(Xp,A,M,C)
plt.plot(Xo[0],Xp,'g-',label=r'$Different \Sigma$')

plt.legend(bbox_to_anchor=(0.01, 0.99), loc=2, borderaxespad=0.)
plt.xlabel(r'$X_{0}$')
plt.ylabel(r'$X_{1}$')
plt.show()
```

executed in 525ms, finished 21:38:42 2018-02-12



Approach 2 : Machine learning

Now, $X =< X_1, X_0 >$ for decision boundary we can define $X_1 = aX_0^2 + bX_0 + c$ here c = C.

We can define X as $X = QX'$ where $X' = \begin{bmatrix} X_0^2 & 0 \\ X_0 & 0 \\ 1 & X_0 \end{bmatrix}$ and $Q = \begin{bmatrix} a & b & 1 \end{bmatrix}$

let's say $H_Q(X) = X(\Sigma_1^{-1} - \Sigma_0^{-1})X^T - 2X(\Sigma_1^{-1}(\mu_1^T) - \Sigma_0^{-1}(\mu_0^T)) + C$ or $H_Q(X) = QX'(\Sigma_1^{-1} - \Sigma_0^{-1})QX'^T - 2QX'(\Sigma_1^{-1}(\mu_1^T) - \Sigma_0^{-1}(\mu_0^T)) + C$ Now we know that for each X, $H_Q(X)$ should be 0 so we use machine learning to estimate parameters $Q < a, b, 1 >$ Where

$J(Q) = \sum_{i=1}^{m} (Y^i - H_Q X^i)^2$ where $Y^i = 0 \, \forall \, i$

$$\nabla_Q J(Q) = -\sum_{i=1}^{m} H_Q(X^i)(2QX'(\Sigma_1^{-1} - \Sigma_0^{-1}) + 2X'(\Sigma_1^{-1}(\mu_1^T) - \Sigma_0^{-1}(\mu_0^T))) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Update rule is $Q^{t+1} = Q^t - \eta \nabla_Q J(Q^t)$

Note: to see how gardient descend is working use the interactive mode by replacing

```
theta = train(m,X2,X1,C,100,flag=False)
```

with

```
theta = train(m,X2,X1,C,100,flag=True)
```

In [76]:

```
Cxinv1 = np.linalg.pinv(Cx1)
Cxinv0 = np.linalg.pinv(Cx0)
C = np.dot(mu_1, np.dot(Cxinv1, mu_1.T)) - np.dot(mu_0, np.dot(Cxinv0, mu_0.T))+np.
M = -2*(np.dot(Cxinv1,mu_1.T)-np.dot(Cxinv0,mu_0.T))
A = Cxinv1-Cxinv0
Xo = ((-1*(X[:,1]*M[1,0])+C)/(M[0,0]))[0]

X1 = X[:,1]
X2 = X[:,1]**2
_C = np.ones(X[:,1].shape)*C
_0 = np.zeros(X[:,1].shape)

XX0=np.vstack((np.vstack((X2,X1)),_C))
XX1=np.vstack((np.vstack((_0,_0)),X1))

m = X.shape[0]
def data_prep(XX0,XX1,m):
    data = []
    for i in range(m):
        data.append(np.vstack((XX0[:,i],XX1[:,i])).T)
    return np.asarray(data)

data = data_prep(XX0,XX1,m)

def loopdot(data,matrix,m,pow):
    data_loop = []
    if pow == 2:
        for i in range(m):
            data_loop.append(np.dot(data[i,:,:],np.dot(matrix,data[i,:,:].T)))
        return np.asarray(data_loop)
    if pow == 1:
        for i in range(m):
            data_loop.append(np.dot(data[i,:,:],matrix))
    return np.asarray(data_loop)

X2 = loopdot(data,A,m,2)
X1 = loopdot(data,M,m,1)

def train(m,X2,X1,C,iter,flag):
    def plot_model(theta,X,C,error,x):
        plt.figure(3)
        sp1=plt.subplot(121)
        sp1.cla()
        X0 = X[Y[:,0] == 'Canada']
        X1 = X[Y[:,0] == 'Alaska']
        sp1.set_title("Data",fontsize=10,y=1.08)
        sp1.scatter(X0[:,0],X0[:,1],c='b',marker='o',label='Canada')
        sp1.scatter(X1[:,0],X1[:,1],c='r',marker='^',label='Alaska')
        minX = int(np.min(X[:,1]))-1
        maxX = int(np.max(X[:,1]))+1
        X1 = np.asarray([x for x in range(minX,maxX)])
        Xo = theta[0,0]*X1**2+theta[1,0]*X1+C
        sp1.plot(Xo[0],X1,'g-',label=r'$Same \Sigma$')
        sp1.legend(bbox_to_anchor=(0, 0.99), loc=2, borderaxespad=0.)
        sp1.set_xlabel(r'$X_{0}$')
        sp1.set_ylabel(r'$X_{1}$')

        sp2=plt.subplot(122)
        sp2.set_title("Error vs Epoch",fontsize=10,y=1.08)
```

```
        sp2.scatter(x,error[0,0],c='g',marker='o')
        sp2.set_xlabel(r'$Epoch$')
        sp2.set_ylabel(r'$J(\theta)$')

        plt.draw()
        plt.pause(0.01)

    def dJ(theta, m, X2, X1, C):
        _dJ = np.zeros((1,3),np.float32)
        for i in range(m):
            _dJ += (0 - 1 *
                  (np.dot(theta.T, np.dot(X2[i, :, :], theta)
                          ) + np.dot(theta.T, X1[i, :, :]) + C)) * (
                              -1 * (2*np.dot(theta.T, X2[i, :, :]) + X1[i, :, :].T
        return _dJ.T/m

    def error(theta, m, X2, X1, C):
        _e=0
        for i in range(m):
            _e += (0 - 1 *
               (np.dot(theta.T, np.dot(X2[i, :, :], theta)
                       ) + np.dot(theta.T, X1[i, :, :]) + C))**2
        return _e/m

    theta = np.zeros((3,1),np.float32)
    start_time = time()
    for x in range(iter):
        theta[2,0]=1
        theta= theta - 0.0000000001*dJ(theta,m,X2,X1,C)
        if (time()-start_time >= 0.02):
            if flag:
                e = error(theta,m,X2,X1,C)
                plot_model(theta,X,C,e,x)
            start_time=time()
    return theta
theta = train(m,X2,X1,C,100,flag=False)
# print('%s = '%('\u0398'))
# print(theta)
```

executed in 1.39s, finished 20:55:30 2018-02-12

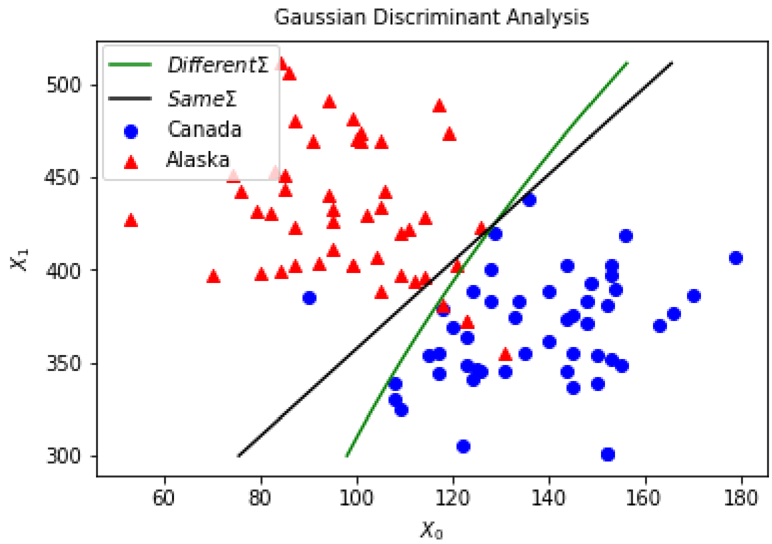
Estimate value for Q = [3.4023385e-04 6.6526326e-07 1.0000000e+00]

Plotting data and learned model for cases $\Sigma_0 = \Sigma_1$ (Same Σ) and $\Sigma_0 \neq \Sigma_1$ (Different Σ)

In [77]:

```
plt.figure(4)
plt.cla()
X0 = X[Y[:,0] == 'Canada']
X1 = X[Y[:,0] == 'Alaska']
plt.title("Gaussian Discriminant Analysis",fontsize=10,y=1.008)
plt.scatter(X0[:,0],X0[:,1],c='b',marker='o',label='Canada')
plt.scatter(X1[:,0],X1[:,1],c='r',marker='^',label='Alaska')

minX = int(np.min(X[:,1]))-1
maxX = int(np.max(X[:,1]))+1

Xp = np.asarray([x for x in range(minX,maxX)])

Xo = theta[0,0]*Xp**2+theta[1,0]*Xp+C
plt.plot(Xo[0],Xp,'g-',label=r'$Different \Sigma$')

Xo = ((-1*(Xp*M_same[1,0])+C_same)/(M_same[0,0]))
plt.plot(Xo[0],Xp,'k-',label=r'$Same \Sigma$')

plt.legend(bbox_to_anchor=(0.01, 0.99), loc=2, borderaxespad=0.)
plt.xlabel(r'$X_{0}$')
plt.ylabel(r'$X_{1}$')
plt.show()
```

executed in 473ms, finished 20:55:31 2018-02-12



Note: Quadratic boundary is separating two class better than linear boundary