Probabilistic Graphical Models: Homework 1

Mohamed N'AITN'BARK

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import pylab
from numpy import linalg
```

In [2]:

```
%matplotlib inline
```

I - Learning in discrete graphical models

Maximum likelihood for the r.a. z:

Let's code our $(z_m)_{m\in[1,M]}$ by vectors of dimension M such that $\forall m, i : z_m^{(i)} = \delta_{m,i}$ where δ is the Kronecker symbol and $z_m^{(i)}$ the ith components of the vector z_m.

Then we can write $P(z) = \prod_{m\in[1,M]} \pi_m^{z^{(m)}}$ and then our likelihood function is: $L(z_1,..,z_n,\pi) = \prod_{i\in[1,N]} \prod_{m\in[1,M]} \pi_m^{z_i^{(m)}}$. Hence the log-likelihood is $l(z_1,..,z_n,\pi) = \sum_{i\in[1,N]} \sum_{m\in[1,M]} z_i^{(m)} log(\pi_m)$. Then we can formulate the maximum likelihood estimator π^{MV} as the solution of the problem:

$$argmax_\pi \sum_{i\in[1,N]} \sum_{m\in[1,M]} z_i^{(m)} log(\pi_m)$$

$$s.t. \forall m \in [1,M] : \pi_m \geq 0 \; and \sum_{m\in[1,M]} \pi_m = 1$$

$$l(z_1,..,z_n,\pi) = \sum_{i\in[1,N]} \sum_{m\in[1,M]} z_i^{(m)} log(\pi_m) = \sum_{m\in[1,M]} (\sum_{i\in[1,n]} z_i^{(m)}) log(\pi_m) = \sum_{m\in[1,M]} n_m log(\pi_m)$$

Where $n_m = card\{i | z_i^m = 1\}$
Then the lagrangian of our problem is $L(\pi,\lambda) = -\sum_{m\in[1,M]} n_m log(\pi_m) + \lambda(\sum_{k\in[1,K]} \pi_k - 1)$
$L(\pi,\lambda)$ is a convex function, and for example $\frac{1}{K}$ **1** $\in]0;+\infty[^K \cap \{\pi | sum_{k\in[1,K]} pi_k = 1\}$ then by Slater's constraint qualification lemma:
$\max_\pi l(\pi) = \max_\lambda \min_\pi L(\pi,\lambda)$
$\frac{\partial L}{\partial \pi_m} = -\frac{\pi_m}{n_m} + \lambda \Rightarrow \pi_m = \frac{n_m}{\lambda}$
The constraint $\sum_{m\in[1,M]} \pi_m = 1$ gives $\lambda = n$ therefore:

$$\forall m \in [1,M] : \pi_m = \frac{n_m}{n}$$

Maximum likelihood for the r.a. x:
An analogous derivation as below shows that our log-likelihood is given by: $l(x_1,..,x_n | z_1,...,z_n,\theta) = \sum_{i\in[1,n]} \sum_{m\in[1,M]} \sum_{k\in[1,K]} x_i^{(k)} z_i^{(m)} log(\theta_m^k)$. Again we can write the loglikelihood in the form: $l(x_1,..,x_n | z_1,...,z_n,\theta) = \sum_{k\in[1,K]} \sum_{m\in[1,M]} (\sum_{i\in[1,n]} x_i^{(k)} z_i^{(m)}) log(\theta_m^k) = \sum_{k\in[1,K]} \sum_{m\in[1,M]} n_m^k log(\theta_m^k)$.
Where $n_m^k = \sum_{i\in[1,n]} x_i^{(k)} z_i^{(m)}$.
This is the same problem as above and by using the same arguments as in the case of z we derive the MV estimator ($n_m = \sum_{k\in[1,K]} n_m^k$):

$$\theta_m^k = \frac{n_m^k}{n_m}$$

II - Linear classification

1 - Generative model (LDA)

** (a) **

The probability of having a realisation (x, y) is $p(x, y; \pi; \mu) = p(x|y; \pi; \mu) = \prod_{k\in[1,K]} (\pi_k f(x, \mu_k))^{y^{(k)}}$ where $f(x, \mu_k)$ is the normal density function $N(\mu_k, \Sigma)$.
Then the log-likelihood of our model is given by:

$$l(\pi, \mu) = \sum_{n\in[1,N]} \sum_{k\in[1,K]} y_n^{(k)} [-\frac{1}{2} (x_n - \mu_k)^T \Sigma^{-1} (x_n - \mu_k) + log(\pi_k)] - Nlog(2\pi) - \frac{N}{2} log(\det \Sigma)$$

$$s.t \sum_{k\in[1,K]} \pi_k = 1$$

The lagrangian of this problem is
$L(\pi, \mu, \lambda) = -\sum_{n\in[1,N]} \sum_{k\in[1,K]} y_n^{(k)} [-\frac{1}{2} (x_n - \mu_k)^T \Sigma^{-1} (x_n - \mu_k) + log(\pi_k)] + \lambda(\sum_{k\in[1,K]} \pi_k - 1)$.
The lagrangian is clearly a convex function and the condition of the Slater's constraint lemma is obviously verified, so we can write:

$$\max_{\pi,\mu} l(\pi, \mu) = \max_{\lambda} \min_{\pi,\mu} L(\pi, \mu, \lambda)$$

$\forall k \in [1, K] : \frac{\partial L}{\partial \pi_k} = \frac{\sum_{n\in[1,N]} y_n^{(k)}}{2\pi_k} + \lambda = 0 \Rightarrow \pi_k = \frac{\sum_{n\in[1,N]} y_n^{(k)}}{2\lambda}$
Since $\sum_{k\in[1,K]} \pi_k = 1$ we get $\lambda = \frac{N}{2}$ and:

$$\forall k \in [1, K] : \pi_k = \frac{n_k}{N}$$

$$n_k = \sum_{n\in[1,N]} y_n^{(k)}$$

$\forall k \in [1, K] : \frac{\partial L}{\partial \mu_k} = \sum_{n\in[1,N]} y_n^{(k)} (x_n - \mu_k) = 0 \Rightarrow \mu_k = \frac{\sum_{n\in[1,N]} y_n^{(k)} x_n}{N}$

** (b) **

In the case of y Bernoulli r.a. :
Using Bayes formula:

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x|y = 1)P(y = 1) + P(x|y = 2)P(y = 2)}$$

i.e.

$$P(y = 1|x) = \frac{\pi_1 \exp -(\frac{(x-\mu_1)^T \Sigma^{-1} (x-\mu_1)}{2})}{\pi_1 \exp -(\frac{(x-\mu_1)^T \Sigma^{-1} (x-\mu_1)}{2}) + \pi_2 \exp -(\frac{(x-\mu_2)^T \Sigma^{-1} (x-\mu_2)}{2})} = \frac{\frac{\pi_1}{\pi_2} \exp (\beta x + \alpha)}{\frac{\pi_1}{\pi_2} \exp (\beta x + \alpha) + 1}$$

Where $\beta = \Sigma^{-1}(\mu_1 - \mu_2)$ and $\alpha = \frac{1}{2} (\mu_2 \Sigma^{-1} \mu_2 - \mu_1 \Sigma^{-1} \mu_1)$.
This model corresponds to a logistic regression with parameters $\beta' = \beta$ and $\alpha' = \alpha + ln(\frac{\pi_1}{\pi_2})$

** (c) **

In [3]:

```
data = np.loadtxt('./classification_data_HWK1/classificationA.train', delimiter="\t")
```

In [4]:

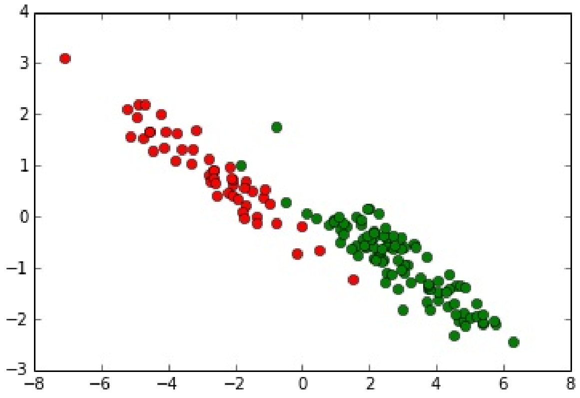
```
class1 = data[data[:,2] == 0][:,(0,1)]
class2 = data[data[:,2] == 1][:,(0,1)]
```

In [5]:

```
pylab.plot(class1[:,0], class1[:,1], 'go', class2[:,0], class2[:,1], 'ro')
```

Out[5]:

```
[<matplotlib.lines.Line2D at 0x7ff358a8bc10>,
 <matplotlib.lines.Line2D at 0x7ff358a8be50>]
```



In [6]:

```
#Estimate the parameters of our model
sigma = class1.T.dot(class1)/len(class1)
mean1 = class1.mean(0)
mean2 = class2.mean(0)
p1 = len(class1)/float(len(data))
p2 = 1-p1
```

In [7]:

```
sigma
```

Out[7]:

```
array([[ 10.71896757,  -3.63945951],
       [ -3.63945951,   1.37479476]])
```

In [8]:

```
mean1
```

Out[8]:

```
array([ 2.89970947, -0.893874  ])
```

In [9]:

```
mean2
```

Out[9]:

```
array([-2.69232004,  0.866042  ])
```

In [10]:

```
p1
```

Out[10]:

```
0.6666666666666666
```

In [11]:

```
#Create a meshgrid to plot the bayesian separator
x = np.linspace(-15,15,200)
y = np.linspace(-15,15,200)
X0,Y0 = np.meshgrid(x, y)
```

In [12]:

```
#Get the slope and intercept of the decision boundary (log(P(y=1|x)) = log(0.5))
def lda_train(x1,x2):
    mean1 = x1.mean(0)
    mean2 = x2.mean(0)
    sigma = (x1-mean1).T.dot(x1-mean1)/len(x1)
    inv_sigma = np.linalg.inv(sigma)
    beta_lda = inv_sigma.dot(mean1 - mean2)
    alpha_lda = 0.5*(mean2.T.dot(inv_sigma.dot(mean2)) - mean1.T.dot(inv_sigma.dot(mean1))) + np.log(p1/p2)
    return beta_lda, alpha_lda
```

In [13]:

```
beta_lda, alpha_lda = lda_train(class2,class1)
```

In [14]:

```
beta_lda
```

Out[14]:

```
array([ -9.05944749, -14.53550779])
```

In [15]:

```
alpha_lda
```

Out[15]:

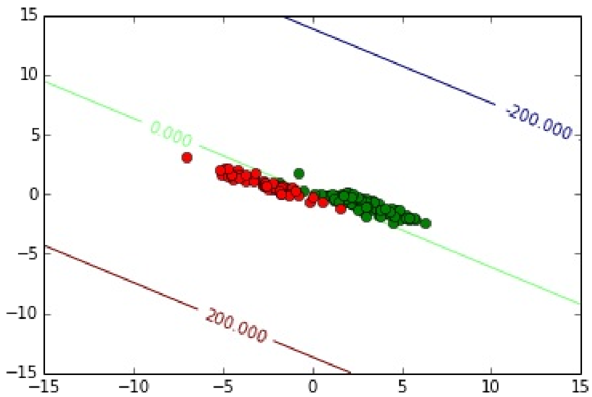
```
1.4302878470837002
```

In [16]:

```
# The decision function values on the meshgrid
Z0 = X0*beta_lda[0] + Y0*beta_lda[1] + alpha_lda
pylab.plot(class1[:,0], class1[:,1], 'go', class2[:,0], class2[:,1], 'ro')
cs = pylab.contour(X0,Y0,Z0,3)
pylab.clabel(cs)
```

Out[16]:

```
<a list of 3 text.Text objects>
```



2 - Logistic regression

In [17]:

```
#Load data
from scipy.special import expit
tab_train = np.loadtxt('./classification_data_HWK1/classificationA.train', delimiter='\t')
```

In [18]:

```
def split_data(data):
    X = data[:,(0,1)]
    X = np.append(X, np.ones((len(X),1)),1)
    Y = data[:,2]
    return X,Y
```

In [19]:

```
# Separate the features from the target variables
X,Y = split_data(tab_train)
```

In [20]:

```
#Implement the Newton Ralphson algorithm
def logreg_train(X,Y):
    #Init
    beta_logreg = (0,0,0)
    threshold = 0.001
    likelihood = []
    #Loop
    while True:
        likelihood.append((Y*np.log(expit(X.dot(beta_logreg))) +(1-Y)*np.log(1-expit(X.dot(beta_logreg)))).
sum())
        beta_prev = beta_logreg
        J = X.T.dot(Y - expit(X.dot(beta_logreg)))
        R = np.diag( expit(X.dot(beta_logreg))*(1 - expit(X.dot(beta_logreg))) )
        H = -(X.T.dot(R)).dot(X)
        beta_logreg = beta_logreg - linalg.solve(H,J)
        if np.linalg.norm(beta_logreg - beta_prev)/np.linalg.norm(beta_prev) < threshold:
            break
    return beta_logreg, likelihood
```

In [21]:

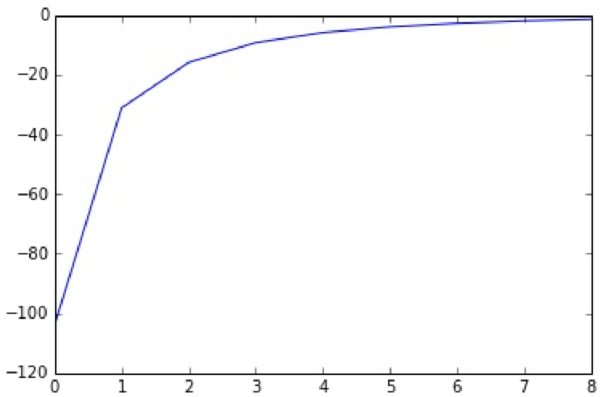
```
beta_logreg, likelihood = logreg_train(X,Y)
```

In [22]:

```
pylab.plot(likelihood)
```

Out[22]:

```
[<matplotlib.lines.Line2D at 0x7ff355c28810>]
```



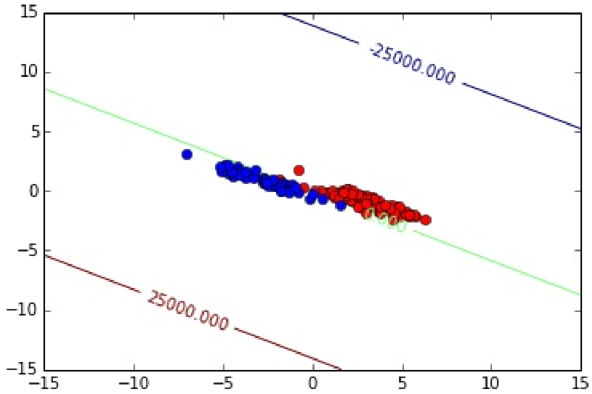
The graph above represents the log likelihood in each iteration of the optimization algorithm

In [23]:

```
Z0 = beta_logreg[0]*X0 + beta_logreg[1]*Y0 + beta_logreg[2]
pylab.plot(tab_train[Y == 0][:,0],tab_train[Y==0][:,1],'ro',tab_train[Y == 1][:,0],tab_train[Y==1][:,1],'bo
')
cs = pylab.contour(X0,Y0,Z0,3)
pylab.clabel(cs)
```

Out[23]:

<a list of 3 text.Text objects>



3 - Linear regression

In [24]:

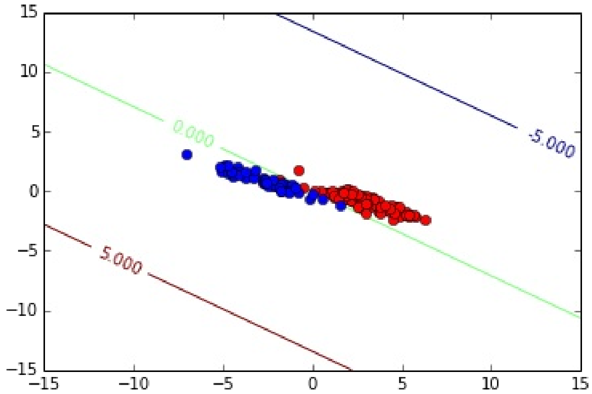
```
# The linear regression model gives the expression of the parameter:
def lin_train(X,Y):
    return linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
```

In [25]:

```
#Now let's draw the data and the line with the equation <x,beta> = 0.5
beta_linreg = lin_train(X,Y)
Z0 = beta_linreg[0]*X0 + beta_linreg[1]*Y0 + beta_linreg[2] - 0.5
pylab.plot(tab_train[Y == 0][:,0],tab_train[Y==0][:,1],'ro',tab_train[Y == 1][:,0],tab_train[Y==1][:,1],'bo
')
cs = pylab.contour(X0,Y0,Z0,3)
pylab.clabel(cs)
```

Out[25]:

<a list of 3 text.Text objects>



4 - Models performance

In [26]:

```
# Let's define a function that computes misclafication error for linear models
def get_err(Y,X,beta,threshold):
    y_pred = (X.dot(beta) >= threshold)
    return np.mean(Y != y_pred)
```

(a) Performance of the implemented models on classifiactionA data set.

In [27]:

```
# Loading test set
testA = np.loadtxt('./classification_data_HWK1/classificationA.test', delimiter='\t')
X_test, Y_test = split_data(testA)
```

In [28]:

```
d = pd.DataFrame(data=np.zeros((6,4)), columns=['Set','lda','log_reg', 'lin_reg'])
d['Set'] = ['trainA','trainB','trainC','testA','testB','testC']
d = d.set_index('Set')
```

In [29]:

```
# Error on training set
#LDA:
beta = np.array([beta_lda[0],beta_lda[1],alpha_lda])
d.ix['trainA','lda'] = get_err(Y,X,beta,0)
#Logistic regression
d.ix['trainA','log_reg'] = get_err(Y,X,beta_logreg,0)
# Linear regression
d.ix['trainA','lin_reg'] = get_err(Y,X,beta_linreg,0.5)

# Error on test set
d.ix['testA','lda'] = get_err(Y_test,X_test,beta,0)
#Logistic regression
d.ix['testA','log_reg'] = get_err(Y_test,X_test,beta_logreg,0)
# Linear regression
d.ix['testA','lin_reg'] = get_err(Y_test,X_test,beta_linreg,0.5)
```

(b) Now let's see how the three models perform on other data sets

In [30]:

```
# Loading the data sets
train_B = np.loadtxt('./classification_data_HWK1/classificationB.train', delimiter='\t')
test_B = np.loadtxt('./classification_data_HWK1/classificationB.test', delimiter='\t')
train_C = np.loadtxt('./classification_data_HWK1/classificationC.train', delimiter='\t')
test_C = np.loadtxt('./classification_data_HWK1/classificationC.test', delimiter='\t')
X_b,Y_b = split_data(train_B)
X_c,Y_c = split_data(train_C)
X_test_b,Y_test_b = split_data(test_B)
X_test_c,Y_test_c = split_data(test_C)
```

In [31]:

```
#Training the three models
#LDA
b,a = lda_train(X_b[Y_b==1][:,(0,1)],X_b[Y_b==0][:,(0,1)])
beta_b_lda = np.array([b[0],b[1],a])
b,a = lda_train(X_c[Y_c==1][:,(0,1)],X_c[Y_c==0][:,(0,1)])
beta_c_lda = np.array([b[0],b[1],a])
#Logistic regression
beta_b_log = logreg_train(X_b,Y_b)[0]
beta_c_log = logreg_train(X_c,Y_c)[0]
#Linear regression
beta_b_lin = lin_train(X_b,Y_b)
beta_c_lin = lin_train(X_c,Y_c)
```

In [32]:

```
#Estimating misclassification error
#LDA:
d.ix['trainB','lda'] = get_err(Y_b,X_b,beta_b_lda,0)
d.ix['testB','lda'] = get_err(Y_test_b,X_test_b,beta_b_lda,0)
d.ix['trainC','lda'] = get_err(Y_b,X_b,beta_b_lda,0)
d.ix['testC','lda'] = get_err(Y_test_b,X_test_b,beta_b_lda,0)

#Logistic regression
d.ix['trainB','log_reg'] = get_err(Y_b,X_b,beta_b_log,0)
d.ix['testB','log_reg'] = get_err(Y_test_b,X_test_b,beta_b_log,0)
d.ix['trainC','log_reg'] = get_err(Y_c,X_c,beta_c_log,0)
d.ix['testC','log_reg'] = get_err(Y_test_c,X_test_c,beta_c_log,0)

# Linear regression
d.ix['trainB','lin_reg'] = get_err(Y_b,X_b,beta_b_lin,0.5)
d.ix['testB','lin_reg'] = get_err(Y_test_b,X_test_b,beta_b_lin,0.5)
d.ix['trainC','lin_reg'] = get_err(Y_c,X_c,beta_c_lin,0.5)
d.ix['testC','lin_reg'] = get_err(Y_test_c,X_test_c,beta_c_lin,0.5)
```

In [33]:

```
d
```

Out[33]:

	lda	**log_reg**	**lin_reg**
Set			
trainA	0.0200	0.000000	0.013333
trainB	0.1400	0.020000	0.030000
trainC	0.1400	0.040000	0.055000
testA	0.0360	0.034000	0.020667
testB	0.1525	0.043000	0.041500
testC	0.1525	0.022667	0.042333

(d) Comments

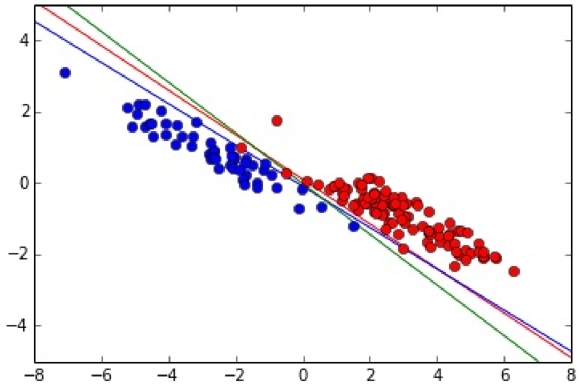
We see that the three models performs very well on the train sets, but comparing the results reveals that LDA is not suited to the data sets B and C. In the test set, logistic and linear regression are outperforming LDA.

In [34]:

```
x = np.linspace(-8,8,200)
y = np.linspace(-5,5,200)
X0,Y0 = np.meshgrid(x, y)
Z0 = X0*beta_lda[0] + Y0*beta_lda[1] + alpha_lda
Z1 = beta_logreg[0]*X0 + beta_logreg[1]*Y0 + beta_logreg[2]
Z2 = beta_linreg[0]*X0 + beta_linreg[1]*Y0 + beta_linreg[2] - 0.5
pylab.plot(X[Y==1][:,0],X[Y==1][:,1],'bo',X[Y==0][:,0],X[Y==0][:,1],'ro')
pylab.contour(X0,Y0,Z0,1,colors='r')
pylab.contour(X0,Y0,Z1,1,colors='b')
pylab.contour(X0,Y0,Z2,1,colors='g')
```

Out[34]:

<matplotlib.contour.QuadContourSet instance at 0x7ff355b90050>



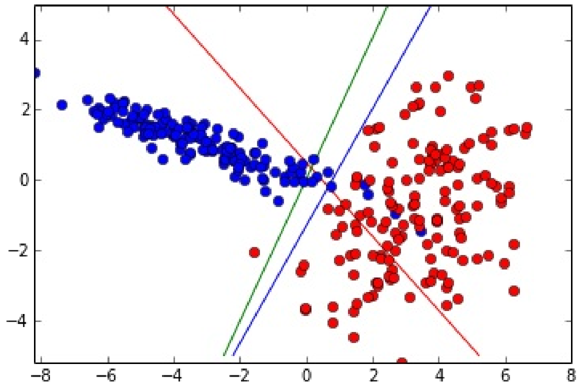
Data set A: LDA separator (red), Logistic regression separator (blue), Linear regression separator (green)

In [35]:

```
x = np.linspace(-8,8,200)
y = np.linspace(-5,5,200)
X0,Y0 = np.meshgrid(x, y)
Z0 = X0*beta_b_lda[0] + Y0*beta_b_lda[1] + beta_b_lda[2]
Z1 = beta_b_log[0]*X0 + beta_b_log[1]*Y0 + beta_b_log[2]
Z2 = beta_b_lin[0]*X0 + beta_b_lin[1]*Y0 + beta_b_lin[2] - 0.5
pylab.plot(X_b[Y_b==1][:,0],X_b[Y_b==1][:,1],'bo',X_b[Y_b==0][:,0],X_b[Y_b==0][:,1],'ro')
pylab.contour(X0,Y0,Z0,1,colors='r')
pylab.contour(X0,Y0,Z1,1,colors='b')
pylab.contour(X0,Y0,Z2,1,colors='g')
```

Out[35]:

<matplotlib.contour.QuadContourSet instance at 0x7ff3559ef440>



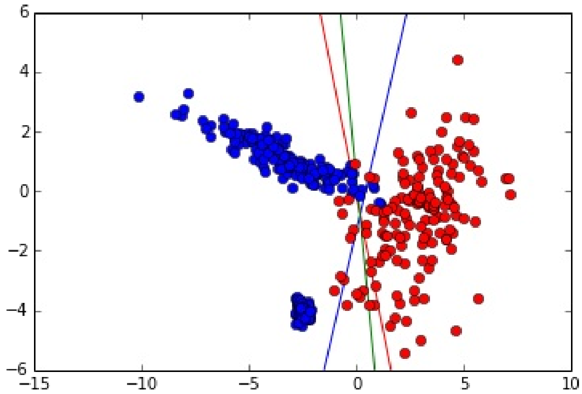
Data set B: The LDA separator classifies poorly the data comparing to the two others. This is due to the fact that LDA assumes that the two clusters have the same covariance.

In [36]:

```
x = np.linspace(-15,10,200)
y = np.linspace(-6,6,200)
X0,Y0 = np.meshgrid(x, y)
Z0 = X0*beta_c_lda[0] + Y0*beta_c_lda[1] + beta_c_lda[2]
Z1 = beta_c_log[0]*X0 + beta_c_log[1]*Y0 + beta_c_log[2]
Z2 = beta_c_lin[0]*X0 + beta_c_lin[1]*Y0 + beta_c_lin[2] - 0.5
pylab.plot(X_c[Y_c==1][:,0],X_c[Y_c==1][:,1],'bo',X_c[Y_c==0][:,0],X_c[Y_c==0][:,1],'ro')
pylab.contour(X0,Y0,Z0,1,colors='r')
pylab.contour(X0,Y0,Z1,1,colors='b')
pylab.contour(X0,Y0,Z2,1,colors='g')
```

Out[36]:

```
<matplotlib.contour.QuadContourSet instance at 0x7ff355844c68>
```



Data set C: Logisitic and linear regression give better result due to the fact that the goal is to improve directly the classification error, while the LDA fits a model to the data, the deduce the classification, which explains the poor performance in the case of data that doesn't fit to the LDA model.

5 - QDA model

Using bayes formula we derive the expression:

$$P(y=1|x) = \frac{1}{\exp-(x^T \mathrm{A} x + \mathrm{B} x + \alpha) + 1}$$

Where $\mathrm{A} = \frac{1}{2}\left(\Sigma_2^{-1} - \Sigma_1^{-1}\right)$ and $\mathrm{B} = \mu_1^T\Sigma_1^{-1} - \mu_2^T\Sigma_2^{-1}$ and $\alpha = \frac{1}{2}\left(\mu_2^T\Sigma_2^{-1}\mu_2 - \mu_1^T\Sigma_1^{-1}\mu_1\right)$

** (a) **

In [37]:

```
# This function trains a QDA model
def qda_train(X,Y):
    class1 = X[Y == 1]
    class2 = X[Y == 0]
    p1 = np.mean(Y==1)
    mean1 = class1.mean(0)
    mean2 = class2.mean(0)
    sigma1 = (class1-mean1).T.dot(class1-mean1)/len(class1)
    sigma2 = (class2-mean2).T.dot(class2-mean2)/len(class2)
    sigma1_inv = linalg.inv(sigma1)
    sigma2_inv = linalg.inv(sigma2)
    A = 0.5*(sigma2_inv - sigma1_inv)
    B = mean1.T.dot(sigma1_inv) - mean2.T.dot(sigma2_inv)
    alpha = 0.5*(mean2.T.dot(sigma2_inv).dot(mean2) - mean1.T.dot(sigma1_inv).dot(mean1)) + np.log(p1/(1-p1
))
    return A,B,alpha
```

In [38]:

```
# Train QDA model on the trainC dataset
A,B,alpha = qda_train(X_c[:,(0,1)],Y_c)
```

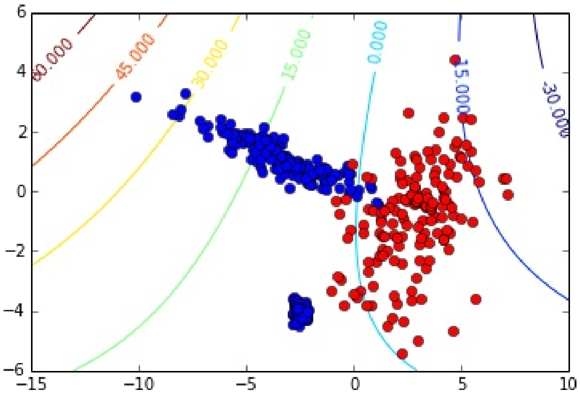
(b) We can now visualize the data with decision boundary

In [39]:

```
Z0 = A[0,0]*(X0**2) + (A[1,0]+A[0,1])*X0*Y0 + A[1,1]*(Y0**2) + B[0]*X0 + B[1]*Y0 + alpha
pylab.plot(X_c[Y_c==1][:,0],X_c[Y_c==1][:,1],'bo',X_c[Y_c==0][:,0],X_c[Y_c==0][:,1],'ro')
cs = pylab.contour(X0,Y0,Z0)
pylab.clabel(cs)
```

Out[39]:

```
<a list of 7 text.Text objects>
```



Data set C

** (c) ** The misclassification error

In [40]:

```
X_c = X_c[:,(0,1)]
Y_c_pred = (np.diag(X_c.dot(A).dot(X_c.T)) + X_c.dot(B) + alpha >= 0)
err_qda = np.mean(Y_c != Y_c_pred)
err_qda
```

Out[40]:

```
0.052499999999999998
```

(d) Comments

The QDA model fits more to the data set C by relaxing the constraint of having the same covariance matrix. But still logistic and linear regression perform better. But we can see that class y = 1 presents two clusters which is clearly not a gaussian distribution proprety, this reduce the performance of QDA. In the other hand logistic and linear regression are model agnostic and hence more robust.