Assignment - Logistic Regression

Year 2017-2018 - Semester II

CCE3501 / CCE3502

Originally devloped by - Adrian Muscat, 2018

Franklyn Sciberras BSc. Computer Science 0441498(m)

A system measures the width and length in centimetres of two fish species, spnott and awrat. The results are given in a csv (comma separated) text file. In this assignment you will develop a linear discriminant function and a logistic regressor that model the data.

We first load the dataset and print the first 10 entries to get a handle on the data. The length is given in the 1st column, the width is given in the second column and the species is given in the third column.

What is the size of the dataset and the class distribution?

Plot a scatter-plot that depicts type_A as a blue dot and type_B as a red dot. As you can see we have to use both input variables, since any variable on its own does not provide enough discrimination power.

Check whether the two input variables are correlated.

We will now split the dataset into two parts. One part is called the training set and the other part is called the test set.

Plot a scatter-plot for the training set and on the same plot add a green line and randomly optimise the gradient and intercept coefficients to minimize the overall classification error.

Repeat above optimisation to balance the classification error in each class.

Code the gradient descent algorithm to minimize the cross-entropy cost function in a logistic regression model

What is the overall classification rate and the class classification rate

CCE3501 only Compare these results with the ones obtained with the random search in task 3.

In [3]:

```
# import useful libraries
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import random
import csv
```

```
# this line plots graphs in line
%matplotlib inline
```

We first load the dataset and print the first 10 entries. The length is given in the 1st column, the width is given in the second column and the species is given in the third column.

In [4]:

```
# DO NOT MODIFY THIS CELL
with open('Fish_Dataset.csv', 'rbU') as f:
    reader = csv.reader(f)
    data_=[]
    for i,row in enumerate(reader):
        if i<2 or i==3:
            if i==0: print row
        else:
            data_.append(row)
data = np.array(data_,dtype=float)
data[0:10]
```

```
['Length', ' Width', ' Class(0=spnott', ' 1=awrat)']
```

Out[4]:

```
array([[ 30.14,  10.27,   1.  ],
       [ 18.28,   7.5 ,   1.  ],
       [ 41.39,  10.04,   0.  ],
       [ 19.19,   7.7 ,   1.  ],
       [ 40.22,  13.07,   0.  ],
       [ 31.63,  14.27,   1.  ],
       [ 37.43,   9.29,   0.  ],
       [ 17.11,   6.8 ,   1.  ],
       [ 39.5 ,  11.47,   0.  ],
       [ 29.22,  13.18,   1.  ]])
```

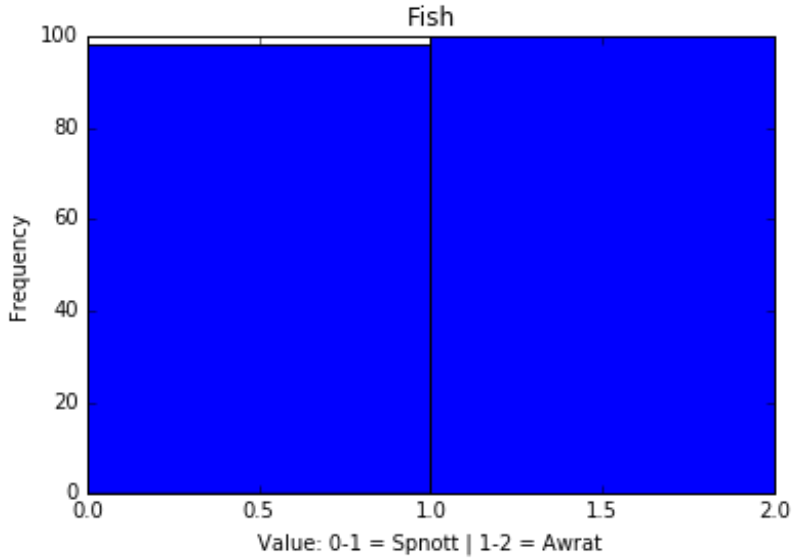
What is the size of the dataset and the class distribution?

In [15]:

```
awrat = 0.0
spnott = 0.0
distribution = []

print "Size of DataSet: ", len(data)

for row in data:
    distribution.append(row[2])
    if row[2] == 1:
        awrat+=1;
    else:
        spnott+=1;

print "Awrat = ", awrat
print "Spnott = ", spnott
```

```
plt.hist(distribution, bins = [0,1,2])
plt.title("Fish")
plt.xlabel("Value: 0-1 = Spnott | 1-2 = Awrat")
plt.ylabel("Frequency")
plt.show()
```

```
Size of DataSet:  198
Awrat =  100.0
Spnott =  98.0
```

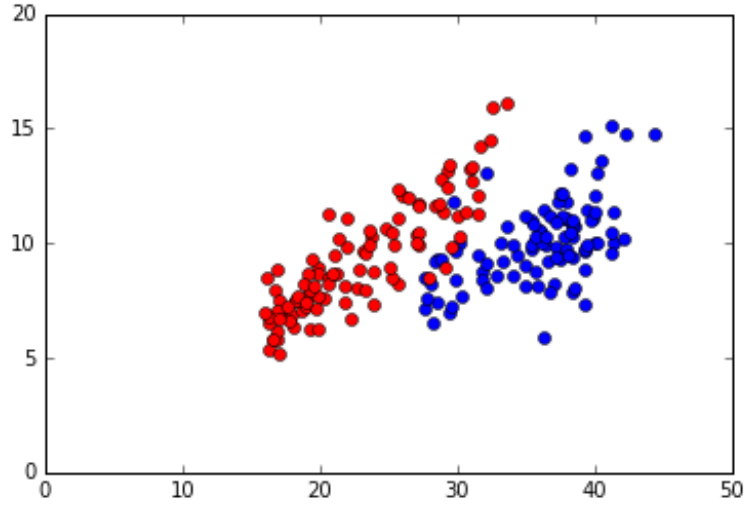


Plot a scatter-plot that depicts spnott as a blue dot and awrat as a red dot. As you will see we have to use both input variables, since any variable on its own does not provide enough discrimination power.

In [39]:

```
def printScatterPlot(limit):
    awrata_feature1 = []
    awrata_feature2 = []

    spnott_feature1 = []
    spnott_feature2 = []

    i = 0

    for row in data:
        if i<=limit:
            if row[2] == 1:
                awrata_feature1.append(row[0])
                awrata_feature2.append(row[1])
            else:
                spnott_feature1.append(row[0])
                spnott_feature2.append(row[1])
            i+=1
        else:
            break

    plt.plot(spnott_feature1, spnott_feature2, 'bo')
    plt.plot(awrata_feature1, awrata_feature2, 'ro')
```

```
    plt.plot(awlata_feature1, awlata_feature2, 'ro')
    plt.axis([0, 50, 0, 20])

printScatterPlot(len(data))
plt.show()
```



Comment on whether the two input variables are correlated.

In [40]:

```
data_first_col = []
data_second_col = []

for row in data:
    data_first_col.append(row[0])
    data_second_col.append(row[1])

corr_level = np.corrcoef(data_first_col,data_second_col)

if(corr_level[0][1]>0.05):
    print "Positively Correlated!"
elif(corr_level[0][1]<-0.05):
    print "Negatively Correlated!"
else:
    print "Weak to No Correlation!"
```

```
Positively Correlated!
```

We will now split the dataset into two parts. One part is called the training set and the other part is called the test set.

In [41]:

```
# DO NOT MODIFY THIS CELL
train_split = int(len(data)*0.7)
print 'Split dataset at ', train_split
d_train=np.array(data[0:train_split,:])
d_test = np.array(data[train_split:,:])
print 'Shape of train = ',np.shape(d_train)
```

```
print 'Shape of test = ',np.shape(d_test)
```

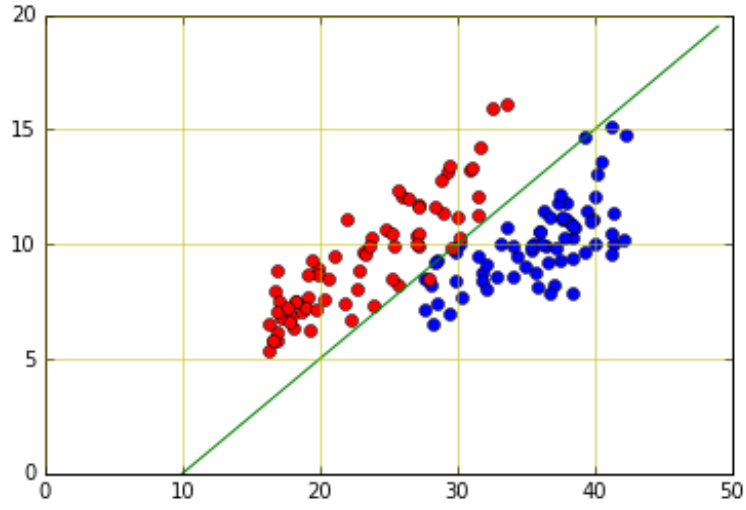
```
Split dataset at  138
Shape of train =  (138, 3)
Shape of test =  (60, 3)
```

Plot a scatter-plot for the training set and on the same plot add a green line and randomly optimise the gradient and intercept coefficients to minimize the overall classification error.Repeat optimisation to maximize the overall classification error.

In [150]:

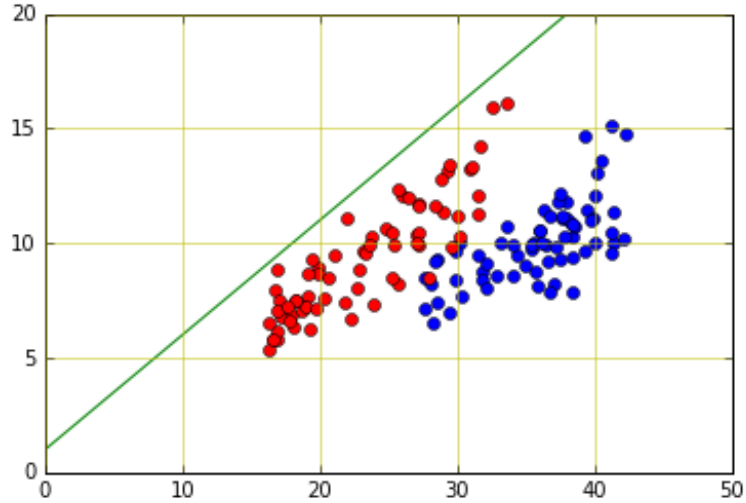
```
x_axis = []

for i in range(0,50):
    x_axis.append(i)

print "Randomly Generated Boundary, To Minimise Classification Error:"
printScatterPlot(train_split)
plt.grid(color='y', linestyle='-', linewidth=0.5)
plt.plot(x_axis, np.add(np.multiply(x_axis, 0.5),-5), color='green')
plt.show()

print "Randomly Generated Boundary, To Maximise Classification Error:"
print "Note that the first graph would have a greater percentage error sinc
e there will be 100 bad"
print "predictions, compared to the 98 bad predictions of the second
graph."
printScatterPlot(train_split)
plt.grid(color='y', linestyle='-', linewidth=0.5)
plt.plot(x_axis, np.add(np.multiply(x_axis, 0.5),1), color='green')
plt.show()
printScatterPlot(train_split)
plt.grid(color='y', linestyle='-', linewidth=0.5)
plt.plot(x_axis, np.add(np.multiply(x_axis, 0.5),-12), color='green')
plt.show()
```

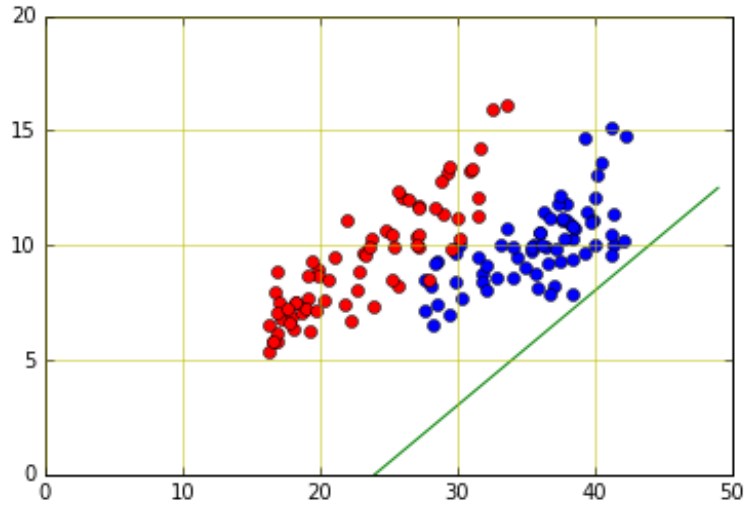
```
Randomly Generated Boundary, To Minimise Classification Error:
```




```
Randomly Generated Boundary, To Maximise Classification Error:
Note that the first graph would have a greater percentage error since there
```

will be 100 bad
predictions, compared to the 98 bad predictions of the second graph.





Develop a function to predict the class given the weights for a linear discriminant

In [119]:

```
def predict_y(W,D):
    predictions = []
    for row in D:
        z = -W[1] - (W[0]*row[0]) + row[1]
        probability = 1.0/(1.0+np.exp(-z))
        if probability > 0.5:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

#print predict_y([0.5,-4],d_test)
#print "-------"
#print d_test
```

Develop a function to calculate the error, given the predicted values and the data set¶

In [95]:

```
def get_error(D,Y):
    error_count = 0
    counter = 0
    for row in D:
        if row[2] != Y[counter]:
            error_count+=1
        counter+=1
    percentage_error = (float(error_count)/float(len(D)))*100.0
    return percentage_error

#print get_error([[0,0,1],[0,0,0],[0,0,0]],[0,0,0])
```

Develop a function to calculate the classification accuracy, given the predicted values and the data set

In [130]:

```
def get_accuracy(D,Y):
    accurate_count = 0
    counter = 0
    for row in D:
        if row[2] == Y[counter]:
            accurate_count+=1
        counter+=1
    percentage_accuracy = (float(accurate_count)/float(len(D)))*100.0
    return percentage_accuracy
```

Use a Monte-Carlo search method (random search) to find the weight vector that minimizes the overall classification error and plot the resulting decision boundary on top of the dataset.

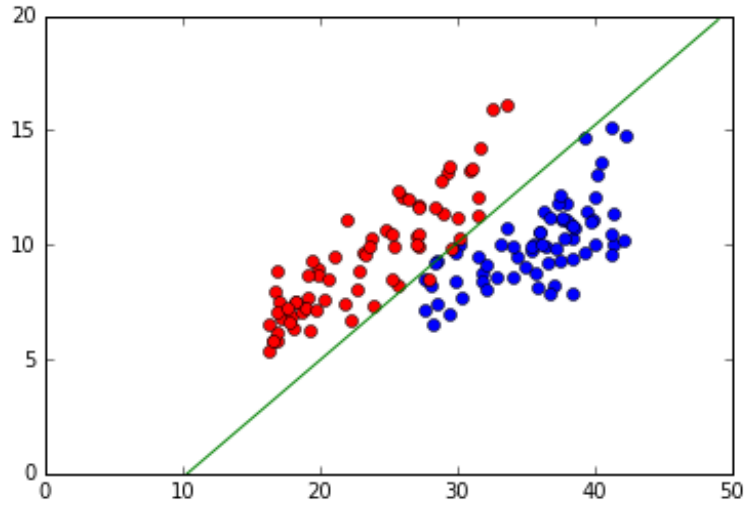
In [97]:

```
def set_X_axis():
    x_axis = []

    for i in range(0,50):
        x_axis.append(i)

    return x_axis

def prod_random_nums():
    gradient = random.uniform(0,1)*1.0
    y_intercept = random.uniform(-6,-3)*1.0
    return [gradient,y_intercept]

#print prod_random_nums()
```

In [165]:

```
def train_MCS():
    printScatterPlot(train_split)
```

```
    iterations = 0

    while True:
        weights = prod_random_nums()
        percentage_error = get_error(d_train,predict_y(weights,d_train))
        if percentage_error < 1.45:
            plt.plot(x_axis, np.add(np.multiply(set_X_axis(), weights[0]),we
ights[1]), color='green')
            plt.show()
            print "Iterations Taken To Complete: ", iterations
            print "Percentage Error: ", percentage_error,"%"
            print "Boundary Equation: y = ",weights[1],"+", weights[0],"x"
            return weights
        iterations+=1

weights_after_training = train_MCS()
```




```
Iterations Taken To Complete:  186
Percentage Error:  1.44927536232 %
Boundary Equation: y =  -5.30936186446 + 0.512916626046 x
```

What is the overall classification accuracy and the class classification accuracy on the test set?

In [166]:

```
#predict outcome of test set using the trained boundary
predictions_test = predict_y(weights_after_training,d_test)

#get accuracy using outcome of test
test_accuracy = get_accuracy(d_test,predictions_test)

#predict outcome of dataset using the trained boundary
predictions_overall = predict_y(weights_after_training,data)

#get accuracy using outcome of whole dataset
overall_accuracy = get_accuracy(data,predictions_overall)

print "Overall Classification Accuracy: ", overall_accuracy, "%"
print "Test Classification Accuracy: ", test_accuracy,"%"
```

```
Overall Classification Accuracy:  97.4747474747 %
Test Classification Accuracy:  95.0 %
```

The remaining questions are only for CCE3501 students

Code the gradient descent algorithm to minimize the cross-entropy cost function in a logistic regression model.

Is there any significant difference between the results with the ones obtained with the random search and the results obtained using gradient descent?.