**Written by: Amin Zollanvari**

**The codes in this document are written using R language.**

**R is an open source software and can be downloaded from**

[**https://www.r-project.org**](https://www.r-project.org)

**1- Basic Concepts:**

**Dataset:**

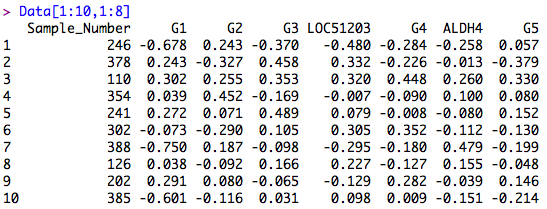
The dataset that we use here has gene expression cancer data. It contains 70 genes (variables) and includes 180 sample points from a group of patients that have had poor prognosis of cancer (we call this class, class 0) and 115 sample points from another group of patients that have had good-prognosis group (lets call this set of patients, class 1).

**Take a look at data:**

First we load the data into R workspace

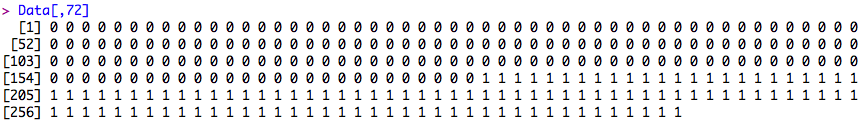
load("Data.Rdata")

The data is in a matrix in which the rows are individuals and the columns are genes). Using the following command (Data[1:10,1:8]), I can see the first 10 rows and 8 columns in this data. The following is a snapshot of the R workplace.



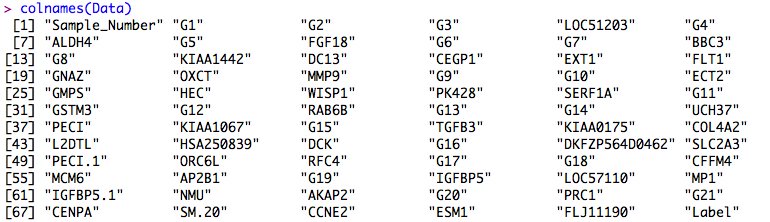
As you see above the first column is the sample number and other columns are labeled by the full/short name for the genes. Each number in these columns (execpt the first one) is representative of the gene expression for that gene (column name) and that sample (sample ID in firsct column).

Now if you type Data[,72], you will see the last column in this data. The last column is a special column in this data that shows a label 0 and 1. If it is 0, this means that the patient has had a poor prognosis and if it is 1, this means that the patient has had good prognosis. Below is the snapshot of what you see in R. Note that there are 180 poor versus 115 good prognoses.

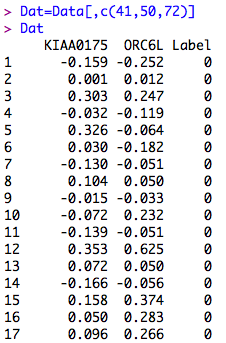
****

**Reducing the dataset for better visualization:**

Since this data has about 70 genes (variables) it is difficult to visualize it. For illustrative purposes we select two genes and we try to demonstrate the classification problem. In this regard, I select two columns that are labeled as KIAA0175 (column 41) and ORC6L (column 50). You can find the names of columns as follows.



Below is how I select this column and assign the new reduced data to variable Dat (below is the snapshot of part of the data for the first 17 samples out of 295)



I intentionally keep the last column of this dataset because it happens that the last column in this data contains the class label information (0 or 1)

Now we can easily visualize the data by the following commands:

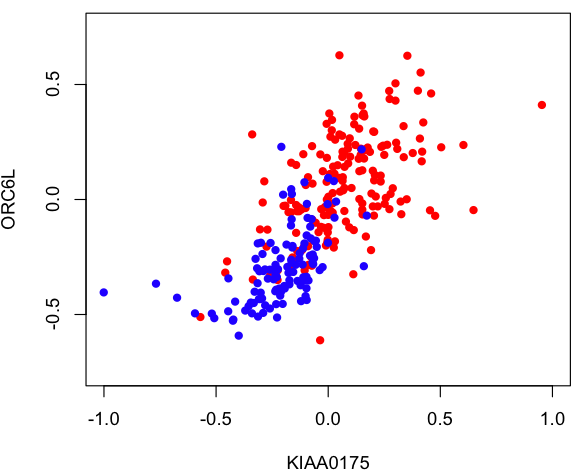
Dat1=Dat[1:180,1:2]

Dat2=Dat[181:295,1:2]

plot(Dat1,xlim=c(-1,1),ylim=c(-0.75,0.75),col="red",pch=16,xlab="KIAA0175",ylab="ORC6L") #plots labled 0 data

points(Dat2,col="blue",pch=16) #plots labled 1 data on the previous plot

The following figure is a scatter plot of ­the data. The red points are label 0 data, the blue points are label 1.



**Classifier:**

Now what is the goal of classification in this example? Imagine in the future you see a patient and you would like to test if this patient has a good prognosis (does not develop cancer) or a poor one (will develop cancer) based on the expression amount of these two genes (KIAA0175 and ORC6L). The goal is to find a mathematical function (this is called training the model) such that the probability of correct classification of this future patient to either 0 or 1 is correct. Now one way to do this, is to use a classifier known as LDA (Linear Discriminant Analysis). There are many more models you can use but LDA is a simple linear and at the same time a powerful model. I have written a function called lda.R that computes the LDA for a 2 dimesnional problem (meaning 2 variables, hee genes). Using this function we can train and plot the LDA model as follows:

# Find LDA and plot

L = lda(Dat1,Dat2)

# in 2D problem

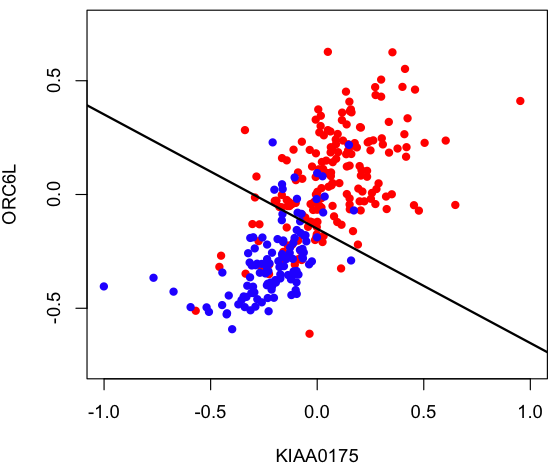
a = L[c(2,3)]

b = L[1]

slope = -a[1]/a[2]

intercept = -b/a[2]

abline(a=intercept,b=slope,lwd=2)



As you see LDA tries to differentiate the red points from blue points by a line. Of course you can imagine there could be many lines but the line that is obtained from LDA formula has slope -1.0187 and intercept -0.5024. You may wonder how the LDA.R function is written.

Lets say in the “training” gene expression data (by “training data” we mean the gene expression data using which we build the model) we have n0 sample points from class 0 (bad prognosis, here n0=180) and n1 sample points from class 1 (good prognosis, here n0=115) and we record these data in



and



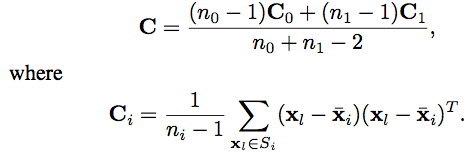
where n= n0+ n1 (so n=295). Therefore, each **xi** is a two dimesnional vector in which the first dimesnion keeps the gene expression values of KIAA0175 and the second dimension keeps the expression values of ORC6L. Now lets say the gene expression values of a future patients for the same genes is recorded in a vector **x**. If the following function of **x** (lets call it WLDA(x)) is larger than 0 then we classify the patient as 0 and if it is less than 0 we classify the patient as 1. This function is called linear discriminant function and is also known as Anderson’s statistic.



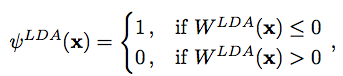
where



are the sampe means of class 0 and 1, and



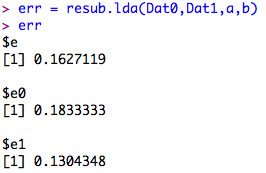
and **C** isthe pooled sample covariance matrix. Now look into file LDA.R to see how these are implemented. Therefore, our LDA classifier is determined by function WLDA(x) and is defined as:



**Error Estimation:**

A critical question after a classifier is designed is how well this classifier performs on future sample points. In other words, what is the probability of misclassifying a future sample point (generalization error). Lets say here we have designed our classifier on the cancer data, and we would like to use this classifier in the future on some data taken from a patient, which we have no idea if this patient has good prognosis (don’t develop cancer) or poor prognosis (will develop cancer). Nevertheless, we expect our trained classifier classifies the patient to either class 0 or 1. However, there is always a probability that the classifier misclassifies the patient for us. This means that it classifies the patient to class 0 while the patient is really from class 1. Or alternatively, classifies the patient to class 1 while the patients belongs really to class 0. These probabilities are basically the “generalization error” (or true error) of the classifier. If we have a set of test data (this is a data that has not been used in training the classifier), true error can be evaluated on test data. Otherwise, we have to estimate the true error using the same training data that is used to train the classifier. One way to estimate this error is using the apparent error or resubstituion estimator. This specific estimator of generalization error is obtained by computing the proportion of errors that the trained classifier makes on the training data.

In our example, this is achieved by using resub.lda function. Here us a snapshot of the workplace.



This means that if the future sample is truly from class 0 (poor prognosis), the classifier with probability 0.183 classify the sample point incorrectly to class 1 (good prognosis) and with probability 1-0.183=0.817 correctly classify the patient as poor prognosis. The last number 0.130 is the probability of misclassification if the future sample point is truly coming from class 1. Now if we weigh both e0 and e1 with prior probability of clases we get e=0.162 (assuming we have mixture sampling).