Chapter 3 k-nearest neighbor and naïve Bayes classifier

The C-Tree in chapter 2 will build a model based on the training dataset as soon as the training dataset is available. It is an example of eager learner. The model is then applied to testing dataset for prediction. An opposite strategy is not to build a model from the training dataset; but make prediction using the training dataset when a testing record is available. It is an example of lazy learner. K-nearest neighbor (knn) and naïve Bayes (NB) classifiers are examples of lazy learner. In this chapter, knn and NB will be briefly introduced.

3.1 K-nearest neighbor (knn)

The idea of knn is relatively simply and intuitive. Given a testing record x, we first compute the k records $t_1,...,t_k$ from the training dataset which are "nearest" to x. These k records are from the training dataset, they should have class labels. Then we predict x to be in class c_m where c_m is the majority class label among these k nearest neighbor (break ties randomly if necessary).

R has a library `class` contains this knn function. To illustrate knn, let us first consider the Iris flower dataset. We first partition the original dataset into 100 records of training dataset and 50 records of testing dataset.

```
> d<-read.csv("iris.csv")                  # read in dataset
> set.seed(123)                            # set random seed
> n<-nrow(d)                               # get sample size
> r<-2/3                                   # set sampling ratio
> id<-sample(1:n,size=round(r*n),replace=F) # generate r*n random integers
> d1<-d[id,]                               # training dataset
> d2<-d[-id,]                              # testing dataset
> dim(d1)                                  # display dimension of d1
[1] 100   5
> dim(d2)                                  # display dimension of d2
[1] 50  5
> names(d)                                 # display names in d
[1] "Sepal_len" "Sepal_wid" "Petal_len" "Petal_wid" "Species"
```

Now we have to load the R library class and apply the knn() function as follow:

```
> library(class)              # load library class
> cl<-factor(d1[,5])          # create a factor object for the class label of d1
> iris.knn<-knn(d1[,1:4],d2[,1:4],cl,k=3)   # apply knn with k=3
```

Note that we have to create a factor object which is the class label of the training dataset d1. To assess the performance of this knn, we produce the classification table as follow:

```
> table(iris.knn,d2$Species)
iris.knn  1  2  3
       1 14  0  0
       2  0 17  1
       3  0  2 16          # error rate = (1+2)/50 = 6%
```

```
d<-read.csv("hmeq1.csv")
(n<-nrow(d))                # get and display sample size
[1] 3067
names(d)
 [1] "BAD"     "LOAN"    "MORTDUE" "VALUE"   "REASON"  "JOB"     "YOJ"
 [8] "DEROG"   "DELINQ"  "CLAGE"   "NINQ"    "CLNO"    "DEBTINC"
```

Note that the 5th and 6th columns (REASON and JOB) are nominal variables and contains characters instead of numeric values. Let us first ignore these two variables and use other variables in knn. We set the random seed and partition the data matrix x into training dataset x1 and testing dataset x2 as before.

```
set.seed(123)                                  # set random seed
r<-2/3                                         # set sampling ratio
id<-sample(1:n,size=round(r*n),replace=F)      # create id for training data
d1<-d[id,]                                     # training data
d2<-d[-id,]                                    # testing data
x1<-d[id,c(1:4,7:13)]                          # exclude columns 5 and 6
x2<-d[-id,c(1:4,7:13)]
```

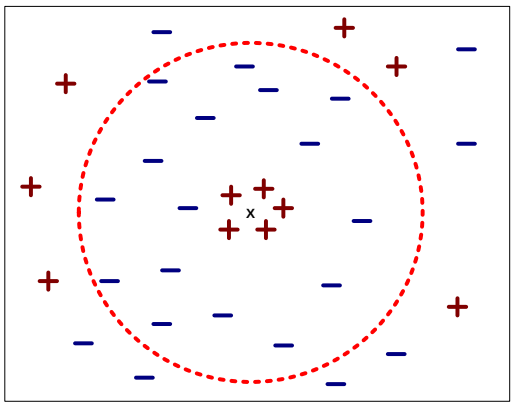
Now we apply the knn with k=3 and produce the misclassification table as follow:

```
c1<-factor(x1[,1])                             # create factor for the label of x1
hmeq.knn<-knn(x1[,2:11],x2[,2:11],c1,k=3)      # apply knn with k=3
table(hmeq.knn,x2[,1])                         # classification table

hmeq.knn   0   1
       0 910  91
       1  11  10          # error rate = (91+11)/1022 = 9.98%
```

Remarks:

1. In our examples, we choose k=3. A natural question is how to choose the value of k. There is no easy answer to this question. If k is too small, knn may be susceptible to over-fitting due to the noises in the training dataset. On the other hand, if k is too large, knn may misclassify the testing records as shown.



2. We predict the class label of the testing record by the **majority voting** of the k nearest neighbors, i.e. $c = \arg\max_v \sum_{i=1}^{k} I(v = t_i)$, where I(.) is the indicator function which equal 1 if the class label of t_i is v and zero otherwise. This means that each k neighbor has equal weight in the voting. One modification is to use **distance weighted voting**. The vote of t_i has weight $w_i = 1/d^2(x,t_i)$, i.e., the weight is inversely proportional to the squared distance of the testing record x to t_i. Then the predict class label of x is $c = \arg\max_v \sum_{i=1}^{k} w_i \ I(v = t_i)$. The distance weight voting also makes knn less sensitive to the choice of k.

3. In computing the distance, the variables may need to standardize if the range of the variables are very different. Otherwise, variables with large range will have large influence in computing the distance. For example, in Iris example, the ranges of the variables are very similar, standardization may not be necessary. However in the HMEQ example, the ranges are very different.

We may need to try several values of k and choose the best result (smallest error rate). We can automated this by writing an improved version `k_nn()` as follow:

```
# improved version of knn
# v is an integer vector containing all the value of k to be tested
# cl0, cl1 are class label for train and test data
# if k is an integer, 1 to k is assumed
# return the k with least error rate

library(class)
k_nn<-function(x0,x1,cl0,cl1,v,l=0,prob=F,use.all=T) {
  err0=1                                  # initialize error rate
  if (length(v)==1) v<-c(1:v)             # change v to an integer vector 1:v
  for (k in v) {
    res<-knn(x0,x1,cl0,k,l,prob,use.all)  # apply knn
    ctab<-table(res,cl1)                  # save c-table
    err<-1-sum(diag(ctab))/sum(ctab)      # compute error rate
    if (err<err0) {                       # update if err<err0
      k0<-k
      res0<-res
      err0<-err
      ctab0<-ctab
    }
    cat("k=",k," error rate=",err,"\n")       # display results
  }
  cat("best k=",k0," error rate=",err0,"\n")  # display best result
  res0                                    # output res0
}
```

Let us try the HMEQ data and standardize these variables using the *stand()* function in chapter 1 before applying `k_nn()`. Let us load this function and apply it to x1 and x2.

```
source("stand.r")     # load the function stand
x<-rbind(x1,x2)       # combine x1 and x2 row-wise
n<-dim(x)[1]          # get row dim of x
n1<-dim(x1)[1]        # get row dim of x1
z<-stand(x)           # standardize x
```

We can easily check that the mean and standard deviation of z is 0 and 1 respectively. Now let us apply the same transformation to the testing dataset d2 and then knn classifier.

```
z1<-z[1:n1,]          # get z1 from z
z2<-z[(n1+1):n,]      # get z2 from z
source( "k_nn.r" )    # load the improved version k_nn function
hmeq.knn<-k_nn(z1[,2:11],z2[,2:11],c1,x2[,1],v=5)   # k_nn using z1,z2 and k=1 to 5

k= 1  error rate= 0.04305284
k= 2  error rate= 0.05088063
k= 3  error rate= 0.05968689
k= 4  error rate= 0.06360078
k= 5  error rate= 0.06360078
best k= 1  error rate= 0.04305284

table(hmeq.knn,x2[,1])                # misclassification table
hmeq.knn   0   1
       0 919  42
       1   2  59                     # error rate = (2+42)/1022 = 4.31%
```

Next we will try the *scale.con()* and *scale.dum()* in the Chapter 1.

```
source("scale.r")                 # load scale function
z1<-scale.con(d1[,-c(1,5,6)])     # transform continuous or ordinal var.
z2<-scale.con(d2[,-c(1,5,6)])
```

```
w1<-scale.dum(d1[,5:6])                    # transform binary or categorical var
w2<-scale.dum(d2[,5:6])
z1<-cbind(z1,w1)                           # combine z and w column-wise
z2<-cbind(z2,w2)
hmeq.knn<-k_nn(z1,z2,c1,x2[,1],v=5)    # knn using z1 ,z2 and k=1 to 5

k= 1  error rate= 0.05283757
k= 2  error rate= 0.06360078
k= 3  error rate= 0.06457926
k= 4  error rate= 0.0704501
k= 5  error rate= 0.07240705
best k= 1  error rate= 0.05283757

table(hmeq.knn,x2[,1])            # misclassification table
hmeq.knn   0   1
       0 915  48
       1   6  53                 # error rate = (6+48)/1022 = 5.28%
```

Finally, we try *k_nn()* on Titanic dataset. Recall that all variables in *titanic.csv* are nominal, we use *scale.dum()* before applying *knn()*.

```
d<-read.csv("titanic.csv")     # read in data
set.seed(12345)                # set random seed
n<-nrow(d)                     # get sample size
r<-0.9                         # set sampling ratio
id<-sample(1:n,round(r*n))     # generate id
d1<-d[id,]                     # training data
d2<-d[-id,]                    # testing data

c1<-factor(d1[,4])             # change target variable to factor
x1<-d1[,1:3]
x2<-d2[,1:3]
w1<-scale.dum(x1)              # using scale.dum() on x1 and x2
w2<-scale.dum(x2)

titan.k_nn<-k_nn(w1,w2,c1,d2[,4],v=5)  # knn with k=1 to 5
k= 1  error rate= 0.2171946
k= 2  error rate= 0.2171946
k= 3  error rate= 0.2171946
k= 4  error rate= 0.2171946
k= 5  error rate= 0.2171946
best k= 1  error rate= 0.2171946

table(titan.knn,d2[,4])        # classification table
titan.knn  no yes
      no  148  46
      yes   1  25              # error rate = (47+1)/220 = 21.82%
```

There are totally 47+1=48 error cases out of 221 testing cases, so the error rate is 48/211=21.72%. Note that the result is same as CTREE in Chapter 2.

3.2 Naïve Bayes classifier

Before we describe the naïve Bayes classifier, let us review the Bayes Theorem in the introductory statistic course.

Recall that the joint and conditional probability of X and Y is related in the following way:

$$\Pr(X,Y) = \Pr(Y \mid X)\Pr(X) = \Pr(X \mid Y)\Pr(Y).$$

This gives the famous **Bayes Theorem**:

$$\Pr(Y \mid X) = \frac{\Pr(X \mid Y)\Pr(Y)}{\Pr(X)}$$

Note that the total probability Pr(X) in the denominator can be written in many different ways depending on different situations. For example:

$$\Pr(X) = \Pr(X \mid Y)\Pr(Y) + \Pr(X \mid \overline{Y})\Pr(\overline{Y}) \quad \text{or}$$

$$\Pr(X) = \sum_{i=1}^{K} \Pr(X \mid Y_i)\Pr(Y_i) \quad \text{where} \quad Y_1,...,Y_K \quad \text{is a partition of the sample space.}$$

Let us illustrate this by the following example:
In a certain factory, machines A, B and C are all producing LCD monitor with defective rate 2%, 1% and 3% respectively. Machine A, B and C produces 35%, 25% and 40% of the total production.

(a) If a LCD monitor is selected at randomly, what is the probability that it is defective?
(b) Given that this LCD monitor is defective, what is the probability that it is produced by machine C?

Answer:

(a) Pr(D)=Pr(D|A)Pr(A)+Pr(D|B)Pr(B)+Pr(D|C)Pr(C)
=(0.35)(0.02)+(0.25)(0.01)+(0.40)(0.03)=0.0215
(b) Pr(C|D)=Pr(D|C)Pr(C)/Pr(D) = (0.40)(0.03)/(0.0215)=0.558

From the above example, we know that the Bayes theorem is useful for classification and prediction. In a general framework, suppose we have a record (from the testing dataset) with attribute $x = (A_1, A_2, ..., A_P)$ and we want to classify it into one of the following class: $(C_1, C_2, ..., C_k)$. Obviously, if we can compute the **posterior probabilities** $\Pr(C_i \mid A_1, ..., A_p)$ $i = 1, .., k$, then we can predict *x* to be in class *j*, where $\Pr(C_j \mid A_1, ..., A_p)$ is maximum among these k probabilities, i.e., $j = \arg\max_i \Pr(C_i \mid A_1, ..., A_p)$.

The key question remains is how to compute $\Pr(C_i \mid A_1, ..., A_p)$ $i = 1, .., k$ from the training dataset. By applying the Bayes Theorem,

$$\Pr(C_i \mid A_1, ..., A_p) = \Pr(A_1, ..., A_p \mid C_i) \Pr(C_i) / \Pr(A_1, ..., A_p)$$

or equivalently, we assign the class label C_j to *x* if $\Pr(A_1, ..., A_p \mid C_j) \Pr(C_j)$ is maximum. Usually $\Pr(C_j)$ can be estimated easily by the proportion of C_j in the training dataset. However, the conditional probability $\Pr(A_1, ..., A_p \mid C_j)$ cannot be estimated easily unless further assumption is imposed. In the naïve Bayes classifier, we assume that attributes $A_1, ..., A_p$ are independent. Therefore,

$$\Pr(A_1, ..., A_p \mid C_j) = \Pr(A_1 \mid C_j) \Pr(A_2 \mid C_j) \cdots \Pr(A_p \mid C_j),$$

and $\Pr(A_i \mid C_j)$ can be estimated by the proportion of record having A_j in the class C_j .

The following example illustrates this. Given the training dataset as follow:

ID	Home Owner (A1)	Marital Status (A2)	Taxable Income (A3)	Default (C)
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

To predict the class label of a testing record *x=(A1=No, A2=Married, A3=$120K)*, we need to compute $Pr(C=No|x) \propto Pr(x|C=No)Pr(C=No)$ and $Pr(C=Yes|x) \propto Pr(x|C=Yes)Pr(C=Yes)$. Note that *Pr(Yes)=0.3* and *Pr(No)=0.7*.

$Pr(x|C=No) \propto Pr(A1=No|C=No)$ x $Pr(A2=Married|C=No)$ x $Pr(A3=120K|C=No)$

$\propto$ 4/7 x 4/7 x $Pr(C3=120K|C=No)$

$Pr(x|C=Yes) \propto Pr(A1=No|C=Yes)$ x $Pr(A2=Married|C=Yes)$ x $Pr(A3=120K|C=Yes)$

$\propto$ 1 x 0 x $Pr(C3=120K|C=Yes)$

To find $Pr(A3=120K|C=No)$ and $Pr(A3=120K|C=Yes)$, we assume that $Pr(A3|C=No)$ is normally distributed with mean *(125+100+…+75)/7=110* and *S.D.=54.54*; $Pr(A3|C=Yes)$ is normally distributed with mean *(95+85+90)/3=90; S.D.=5*.

To estimate $Pr(A3=120K|C=No)$, we use

$$\Pr(120 < A3 < 120+\varepsilon \mid C = No) \approx \frac{\varepsilon}{\sqrt{2\pi}(54.54)} \exp\{(-0.5)[(120-110)/54.54]^2\} = 0.0072\varepsilon .$$

For $Pr(A3=120K|C=Yes)$, we use

$$\Pr(120 < A3 < 120+\varepsilon \mid C = Yes) \approx \frac{\varepsilon}{\sqrt{2\pi}(5)} \exp\{(-0.5)[(120-90)/5]^2\} = 0\varepsilon$$

Therefore, $Pr(C=No|x) \propto Pr(x|C=N0)\,Pr(No) = (4/7) \text{ x } (4/7) \text{ x } (0.0072\varepsilon) \text{ x } (0.7) = 0.0017\,\varepsilon$ while $Pr(C=Yes|x) \propto Pr(x|C=Yes)\,Pr(Yes) = (1) \text{ x } (0) \text{ x } (0\varepsilon) \text{ x } (0.3) = 0$.

Therefore, we predict the class label of x to be No.

Remarks:

In a small training dataset, sometimes the conditional probability is zero. For example, $Pr(A2=Married|C=Yes)=0$. To overcome this problem, we estimate this probability using a Bayesian approach. For example, using Laplace's estimate: $\Pr(A \mid C) = (n_A + 1)/(n + m)$, where *m* is the number of class. In our example, $Pr(A2=Married|C=Yes) = (0+1)/(3+2)=1/5$.

3.3 Examples using Naïve Bayes classifier

The naïve Bayes classifier has been implemented in the library *e1071* called *naiveBayes()*. First, we need to download this *e1071* library from R's website. Make sure your computer is connected to the internet, then choose Packages -> install package(s)… from R's menu and follow the instructions; we should install this library easily. Again we can have online help of this *e1071* library by typing *library(help=e1071)* and online help of *naiveBayes()* by typing *help(naiveBayes)*. Let us illustrate this function by the Iris data.

```
> library(e1071)                          # load library e1071
> d<-read.csv("iris.csv")                 # read in IRIS data
> n<-nrow(d)                              # get sample size
> r<-2/3                                  # set sampling ratio
> set.seed(123)                           # set random seed
> id<-sample(1:n,size=round(r*n),replace=F)
> d1<-d[id,]                              # training data
> d2<-d[-id,]                             # testing data
>
> cl<-factor(d1[,5])                      # define class label
> iris.nb<-naiveBayes(d1[,1:4],cl)        # apply naiveBayes
> pr<-predict(iris.nb,d2[,1:4])           # predict
> table(pr,d2[,5])                        # classification table

pr   1  2  3
  1 14  0  0
  2  0 18  2
  3  0  1 15                  # error rate is (1+2)/50 = 6%
```

Next we try the Titanic dataset:

```
> d<-read.csv("titanic.csv")              # read in Titanic data
> n<-nrow(d)
> r<-0.9
> set.seed(12345)
> id<-sample(1:n,round(r*n))              # default is replace=F
> d1<-d[id,]                              # training data
> d2<-d[-id,]                             # testing data
>
> cl<-factor(d1[,4])                      # define class label
> titan.nb<-naiveBayes(d1[,1:3],cl)       # apply naiveBayes
> pr<-predict(titan.nb,d2[,1:3])          # predict
> table(pr,d2[,4])                        # classification table

pr     no yes
  no  136  40
  yes  13  31                # error rate is (13+41)/220 = 24.1%
```

Finally we try the HMEQ dataset. First note that columns 5 and 6 should be omitted. Column 1 is target variable, columns 8,9, 11 are categorical variables and should be changed to categorical using *factor()*.

```
> d<-read.csv("hmeq1.csv")                # read in HMEQ data
> n<-nrow(d)                              # get sample size
> r<-2/3                                  # set sampling ratio
> set.seed(123)                           # set random seed
> id<-sample(1:n,round(r*n))              # generate id
```

```
> c1<-factor(d[,1])                    # change categorical var to factor
> c8<-factor(d[,8])
> c9<-factor(d[,9])
> c11<-factor(d[,11])
> x2<-d[,c(2,3,4,7,10,12,13)]          # select continuous var.
> x<-cbind(c1,c8,c9,c11,x2)            # combine var. column-wise to form x
>
> d1<-x[id,]                           # training data
> d2<-x[-id,]                          # testing data
> cl<-factor(d1[,1])                   # class label
> hmeq.nb<-naiveBayes(d1[,2:11],cl)    # apply naiveBayes
> pr<-predict(hmeq.nb,d2[,2:11])       # predict
> table(pr,d2[,1])                     # classification table
pr    0   1
  0 912  61
  1   9  40          # error rate = (61+9)/1022 = 6.84%
```

3.4 Measuring Accuracy

When measuring the accuracy of a classification model, the overall accuracy computed from the classification table is not the only measure and can be misleading in some situations. We can have the **Precision, Recall (**also known as **Sensitivity**) and the **F1 score** to measure the accuracy of the model. In medical diagnostic test, we call the result of the test is positive if the test predict the subject have certain disease. For example:

Test \ True	**No Disease**	**Disease**	**row sum**
Negative	*True Negative (TN)*	*False Negative (FN)*	*TN+FN*
Positive	*False Positive (FP)*	*True Positive (TP)*	*FP+TP*
column sum	*TN+FP*	*FN+TP*	Total

The precision, recall and F1 score of the test is defined as follow:

Precision = TP/(FP+TP),

Recall (or Sensitivity) = TP/(TP+FN) = ratio of true positive cases recalled from the data.

$$F1 = \frac{2}{1/\text{Precision} + 1/\text{Recall}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

This F1 score is the harmonic mean of precision and recall. Also the **specificity** of the test is defined as: *Specificity = TN/(TN+FP) = ratio of true negative cases specified by the test.*

The advantage of F1 is illustrated by the following example. Consider a diagnostic test of a rare disease:

Test \ True	**No Disease**	**Disease**	**row sum**
Negative	195	1	196
Postive	2	2	4
column sum	197	3	200

The overall accuracy of the test is (2+195)/200=98.5% and the specificity of the test is 195/197=89.98% is high only due to the disease is rare; most of the correct predictions are come from true negative. However, the precision of the test is only 2/4=50% and the recall (or sensitivity) of the test is 2/3=66.7% and the F1 score is 4/7=57.14%.

3.5 Decision Threshold and Receiver Operating Characteristic (ROC) curve

Making wrong decisions are inevitable. In the HMEQ example, let us define BAD=1 as positive diagnostic test result and BAD=0 as negative diagnostic test.

Customer \ Decision	**Reject**	**Accept**
Bad	True Positive	False Negative (*cost=$c2*)
Good	False Positive (*cost=$c1*)	True Negative

In practice, the cost of false negative is not the same as the cost of false positive. Let $p(x)=Pr\{Bad|x\}$ be the estimated probability of default. Then the decision rule will be: reject the application if $p(x)>c$, where c is a constant known as decision threshold. The question is how to choose a suitable c to account for the cost of misclassification. Note that the expected loss of rejecting a good customer is $c1\ [1-p(x)]$ while expected loss of accepting a bad customer is $c2\ p(x)$. Hence a reasonable decision rule will be: reject the application if $c1[1-p(x)] < c2\ p(x)$, which is equivalent to reject the application if $p(x) > c1/(c1+c2)$.

Sometimes we may not have the cost of misclassification, we can use ROC curve to obtain a reasonable threshold value c. First define **true positive rate(tpr)**=sensitivity=$TP/(TP+FN)$. We can also define **false positive rate(fpr)**=1-specifictiy=$FP/(TN+FP)$. Let us continue with the HMEQ example to illustrate how these tpr and tnr changes with the threshold value c. First note that `prob<-predict(hmeq.nb,d2[,2:11]` only gives the predict value (0 or 1) using c=0.5. We can use `prob<-predict(hmeq.nb,d2[,2:11],type="raw")` to obtain two columns of predicted probabilities of BAD=0 or BAD=1.

```
> prob<-predict(hmeq.nb,d2[,2:11],type="raw")   # prob has 2 columns of prob.
> pr<-(prob[,2]>0.5)      # using c=0.5
> (t<-table(pr,d2[,1]))   # save and display table
pr        0   1
  FALSE 912  61
  TRUE    9  40

> t[2,2]/sum(t[,2])       # true postive rate
[1] 0.3960396
> t[2,1]/sum(t[,1])       # false postive rate
[1] 0.009771987
```

Let us define BAD=1 as positive; BAD=0 as negative and compute tpr and tnr using various threshold value c. Note that c=0.5 gives exactly the classification table on p.9. When c=0.2, both tpr and fpr increase. When c=0.8, both tpr and fpr decrease. In general, tpr and fpr increase as c decrease. In the extreme case when c=1, all cases are predicted as negative; tpr=fpr=0. On the other hand, when c=0, all cases are predicted as positive; tpr=fpr=1.

```
> pr<-(prob[,2]>0.2)       # using c=0.2
> (t<-table(pr,d2[,1]))    # save and display table
pr        0   1
  FALSE 879  51
  TRUE   42  50

> t[2,2]/sum(t[,2])        # true postive rate
[1] 0.4950495
> t[2,1]/sum(t[,1])        # false postive rate
[1] 0.04560261
>
> pr<-(prob[,2]>0.8)       # using c=0.8
> (t<-table(pr,d2[,1]))    # save and display table
pr        0   1
  FALSE 919  76
  TRUE    2  25

> t[2,2]/sum(t[,2])        # true postive rate
[1] 0.2475248
> t[2,1]/sum(t[,1])        # false postive rate
[1] 0.002171553
```

The **Receiver Operating Characteristic(ROC) curve** is the plot of tpr vs fpr for various values of c. We need to install the ROCR library. Once the ROCR is installed, we can use the following commands to produce ROCR curve.

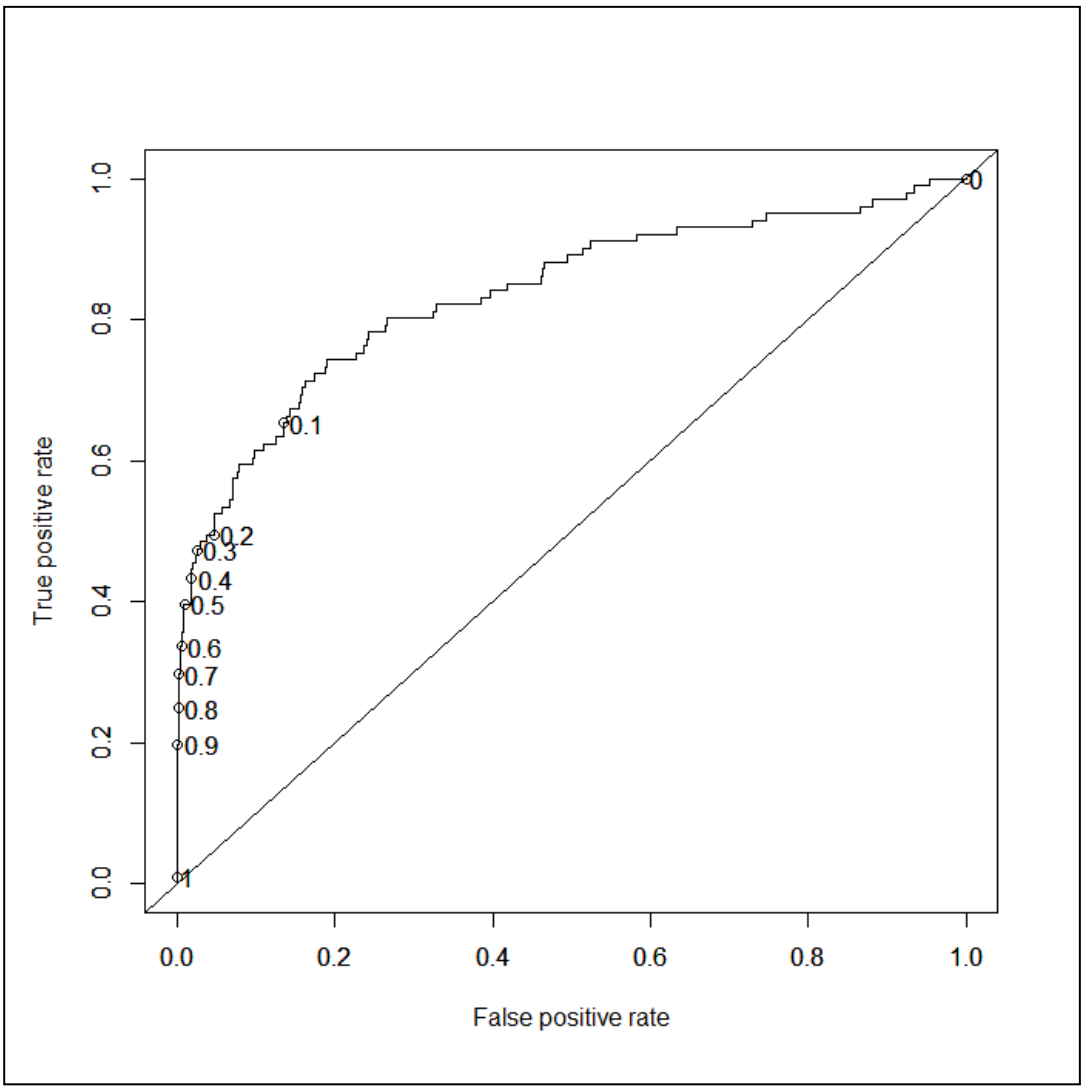
```
# ROCR library
> library(ROCR)                                  # load ROCR library
> roc<-prediction(prob[,2],d2[,1])               # prediction of prob
> perf<-performance(roc,"tpr","fpr")             # performance of roc

> plot(perf,print.cutoffs.at=seq(0,1,0.1),text.adj=c(-0.2,0.5))    # plot with diff c
> abline(a=0,b=1)                                # add in reference line

# Area under ROC curve
> as.numeric(performance(roc,"auc")@y.values)   # output area under curve
[1] 0.8353598
```

In the ROCR library, the functions prediciotn() and performance compute the tpr and fpr for the curve. The function plot will plot the curve tpr vs fpr. The options print.cutoffs will gives the position at various c ranging from 0 to 1 with 0.1 increments. The option text.adj will adjust the numbers so that it will not overlap with the points.

The below ROC curve suggests that c=0.2 give a reasonable high tpr and low fpr. However, if we want a higher tpr, then we can use c=0.1 but paying the price of higher fpr as well.



Finally, we can assess the performance of the model by the area under the ROC curve. As a rule of thumb, 0.9-1=excellent; 0.8-.09=good; 0.7-0.8=fair; 0.6-0.7=poor; 0.50-0.6=fail. In our example, AUC=0.8353598, hence the performance of our model is good.

Reference:

Chapter 5 of Introduction to Data Mining by Tan, Steinbach and Kumar, Addison Wesley.