Lab #3

Purpose: In this lab, we implement the logistic regression and KNN algorithms.

1. Load Data Default.csv to object Default.

```
rm(list = ls())
```

```
Default=read.csv(file.choose(),header=T)
```

Here header is true because the CSV file has column headings in it

```
attach(Default)
```

attach allows you to reference database variables without using the “$” extractor

```
names(Default)
```

```
summary(Default)
```

The *attach* function in R is a powerful tool in base R that allows you to, well, attach a specific database to a search path. In other words, *attach* allows you to reference database variables without using the “$” extractor. For example, without using the *attach* function (using the Iris data set from R):

```
> head(iris)
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1          3.5         1.4          0.2         setosa
2 4.9          3.0         1.4          0.2         setosa
3 4.7          3.2         1.3          0.2         setosa
4 4.6          3.1         1.5          0.2         setosa
5 5.0          3.6         1.4          0.2         setosa
6 5.4          3.9         1.7          0.4         setosa

> #calculating the mean
> mean(iris$Sepal.Length)
[1] 5.843333
```

Now, with the *attach* function:

```
> attach(iris)
> mean(Sepal.Length)
[1] 5.843333
```

2. Logistic Regression

The original data set contains 10000 observations. We first fit model glm.fit1 which includes all three independent variables: student, balance and income, with all the observations. We can use the glm() function.

```
glm.fit1=glm(default~balance+income+student,family="binomial",data=Default)
```

We can use the function summary() to obtain the coefficients, their associated p-values, model AIC and residual deviance. The residual deviance is 1571.5 and the AIC is 1579.5. The function exp() calculates odds ratio of the coefficients.

```
summary(glm.fit1)
```

```
coef(glm.fit1)
```

```
exp(coef(glm.fit1))
```

We next evaluate the prediction accuracy of this model. We use the sample() function to split the original data set into one training set and one test set. We randomly select 80% of the observations for training and the remaining in the test set (Default.test). And save the true value of the testing set in the vector test.truevalue.

```
set.seed(1)
```

```
train=sample(nrow(Default),nrow(Default)*0.8)
```

```
Default.test=Default[-train, ]
```

```
test.truevalue=default[-train]
```

```
train.truevalue=default[train]
```

We fit the data with all three independent variables into a logistic regression model glm.fit2. The subset option in glm() is to fit a regression using only the observations corresponding to the subset.

```
glm.fit2=glm(default~balance+student+income,data=Default,subset=train,family =binomial)
```

```
summary(glm.fit2)
```

```
exp(coef(glm.fit2))
```

We evaluate the performance of the model using the test set (Default.test). We use the function predict() to calculate the predicted probabilities of the default in the test set and store them to glm.pred2. The type="response" option tells R to output probabilities not the logit.

```
glm.probs2=predict(glm.fit2,Default.test, type="response")
```

We convert the predicted probabilities into a binary class label, Yes or No. The following commands create a vector of class predictions based on whether the predicted probability is greater than or less than 0.5.

```
glm.pred2=rep("No",2000)
```

observation

```
glm.pred2[glm.probs2>.5]="Yes"
```

of probability > 0.5 ⇒ Yes

The table() function produce a confusion matrix to determine how many observations were correctly classified. We can find that glm.fit2 correctly predicts 97.25% of default status and misclassify 2.75% in the test set.

```
table(glm.pred2,test.truevalue)
```

the order does matter when the implication is meaningful

```
mean(glm.pred2==test.truevalue)

mean(glm.pred2!=test.truevalue)
```

glm.fit2 correctly predicts 97.5% of default status. $\frac{1930+20}{2000} = 97.5\%$

We next evaluate the 5-fold cross-validation prediction of the model.

```
k=5

folds=sample(1:k,nrow(Default),replace=TRUE)
```

We create a vector to store the accuracy result for each fold. We set the initial values for this vector as zero.

```
accuracy=rep(0,k)

  for(i in 1:k)
   {
    glm.fit3=glm(default~balance+student+income,family="binomial",data=Default[folds!=i,])
    Default.test=Default[folds==i, ]

    glm.probs3 =predict(glm.fit3,Default.test, type="response")
    glm.pred3=rep("No",nrow(Default[folds==i,]))

    glm.pred3[glm.probs3>.5]="yes"

    test.truevalue=default[folds==i]
    accuracy[i]=mean(glm.pred3==test.truevalue)

   }
```

instead of 2000

Then we calculate the average cross-validation accuracy, which is 96.3%.

```
mean(accuracy)
```

3. KNN

We perform KNN using the knn() function, which is part of the class package. We first load the class package.

```
library(class)
```

We first normalize the two quantitative input variables balance and income so that they would be on a comparable scale. The scale() function standardizes the quantitative variables.

```
standardized.balance=scale(balance)
```

```
standardized.income=scale(income)
```

Then we use the function bind() to combine the two standardized variables and the qualitative input variable together.

```
Input.standard=cbind(standardized.balance,standardized.income,student)
```

Instead of vector, we use a matrix

```
accuracy=matrix(0,10,5)
```

rows columns

we have 5 folds

K values

initial value is 0

To use KNN, we need to determine K. We can use 5-fold cross-validation to select the best K from [1,10]. Thus, we first create a matrix to store the accuracy results for five folds and ten different K values. We set the initial values for this matrix as zero.

```
set.seed(2)
```

```
folds=sample(1:5,nrow(Input.standard),replace=TRUE)
```

```
for (j in 1:10)
  {
     for(i in 1:5)
       {
     train.standard=Input.standard[folds!=i,]
     test.standard=Input.standard[folds==i,]
     train.truevalue=default[folds!=i]
     test.truevalue=default[folds==i]
     knn.pred=knn(train.standard,test.standard,train.truevalue,k=j)
     table(knn.pred,test.truevalue)
     accuracy[j,i]=mean(knn.pred==test.truevalue)
       }
  }
```

from 1NN to 10-NN

3x.

3 variables (served for distance)

training dataset

the response variable in training dataset

testing standard

the true value in training dataset

train.standard is the input matrix of the training set and test.standard is the input matrix of the testing set. train.truevalue is the original value of default in the training set and test.truevalue is the true value of default in the testing set. For each observation in the testing set, the knn function can calculate its distance with each observation in the training set based on train.standard and test.standard. Given k=j, it selects the k most nearest neighbors, and use their default values (based on train.truevalue) to predict the default values in the testing set. In the knn function, we specify the number of neighbors in the option "k=j". The output of this loop is the prediction accuracy

Then we calculate the average cross-validation accuracy for each K. The function which.max() tells us that when K=9, it leads to the highest accuracy as 97.17% .Thus, K should be selected as 9.

9-NN is the best

```
cv.accuracy=apply(accuracy,1,mean)
```

```
which.max(cv.accuracy)
```

This concludes lab #3.