Exercise: Logistic regression, k nearest neighbours

In this R exercise, you will know:

- How to get training and test sets
- How to perform logistic regression in R
- How to perform kNN in R
- How to scale data

Don't forget to change your working directory!

Contents

1 R Packages and datasets required

1.1 Use Package class or anything else

- Type the following code to install and activate the package:
  - install.packages("class")
  - library(class)
  - install.packages("caret")
  - library(caret)
  - install.packages("ISLR")
  - library(ISLR)

1.2 Dataset

- Edgar Anderson's Iris Data

```
#This data frame is already contained in the R
#Therefore, no need to read from other source
#Use "?iris" to check the detail about this dataset
```

- Stock Market Data

The Smarket data is part of the ISLR library. This data set consists of percentage returns for the S&P 500 stock index over $1,250$ days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, Lag1 through Lag5. We have also recorded Volume (the number of shares traded on the previous day, in billions), Today (the percentage return on the date in question) and Direction (whether the market was Up or Down on this date).

2 Logistic regression

In this section, we will use the Smarket data. To have a look at some basic properties of the dataset:

```
library(ISLR)
names(Smarket)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
dim(Smarket)
```

```
## [1] 1250    9
```

```
summary(Smarket)
```

```
##       Year           Lag1                Lag2
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500
##  Median :2003   Median : 0.039000   Median : 0.039000
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000
##       Lag3                Lag4                Lag5
##  Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92200
##  1st Qu.:-0.640000   1st Qu.:-0.640000   1st Qu.:-0.64000
##  Median : 0.038500   Median : 0.038500   Median : 0.03850
##  Mean   : 0.001716   Mean   : 0.001636   Mean   : 0.00561
##  3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59700
##  Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73300
##      Volume           Today           Direction
##  Min.   :0.3561   Min.   :-4.922000   Down:602
##  1st Qu.:1.2574   1st Qu.:-0.639500   Up  :648
##  Median :1.4229   Median : 0.038500
##  Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   :3.1525   Max.   : 5.733000
```

The cor() function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. The first command below gives an error message because the Direction variable is qualitative.

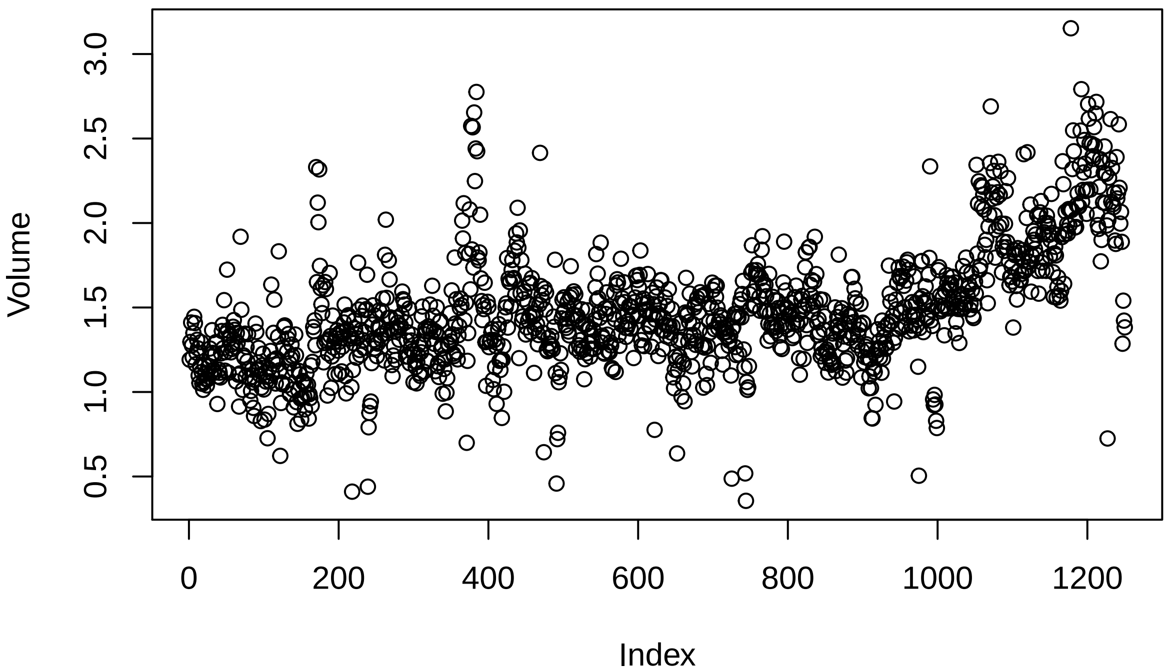
```
#cor(Smarket)
cor(Smarket[, -9])
```

```
##              Year         Lag1         Lag2         Lag3         Lag4
## Year   1.00000000  0.029699649  0.030596422  0.033194581  0.035688718
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402 -0.002985911
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670 -0.010853533
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000 -0.024051036
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036  1.000000000
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686 -0.048414246
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527
##                Lag5      Volume        Today
## Year    0.029787995  0.53900647  0.030095229
## Lag1   -0.005674606  0.04090991 -0.026155045
## Lag2   -0.003557949 -0.04338321 -0.010250033
```

```
## Lag3   -0.018808338 -0.04182369 -0.002447647
## Lag4   -0.027083641 -0.04841425 -0.006899527
## Lag5    1.000000000 -0.02200231 -0.034860083
## Volume -0.022002315  1.00000000  0.014591823
## Today  -0.034860083  0.01459182  1.000000000
```

As one would expect, the correlations between the lag variables and today's returns are close to zero. In other words, there appears to be little correlation between today's returns and previous days' returns. The only substantial correlation is between Year and Volume. By plotting the data we see that Volume is increasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005.

```r
attach(Smarket)
plot(Volume)
```



Next, we will fit a logistic regression model in order to predict Direction using Lag1 through Lag5 and Volume. The glm() function fits generalized linear models, a class of models that includes logistic regression. The syntax of the glm() function is similar to that of lm(), except that we must pass in linear model the argument family = binomial in order to tell R to run a logistic regression rather than some other type of generalized linear model.

```r
glm.fits <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                data = Smarket, family = binomial)

summary(glm.fits)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
```

```
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

The smallest p-value here is associated with Lag1. The negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is less likely to go up today. However, at a value of 0.15, the p-value is still relatively large, and so there is no clear evidence of a real association between Lag1 and Direction.

We use the coef() function in order to access just the coefficients for this fitted model. We can also use the summary() function to access particular aspects of the fitted model, such as the p-values for the coefficients.

```r
coef(glm.fits)
```

```
##  (Intercept)         Lag1         Lag2         Lag3         Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##         Lag5       Volume
##  0.010313068  0.135440659
```

```r
summary(glm.fits)$coef
```

```
##                 Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.126000257 0.24073574 -0.5233966 0.6006983
## Lag1        -0.073073746 0.05016739 -1.4565986 0.1452272
## Lag2        -0.042301344 0.05008605 -0.8445733 0.3983491
## Lag3         0.011085108 0.04993854  0.2219750 0.8243333
## Lag4         0.009358938 0.04997413  0.1872757 0.8514445
## Lag5         0.010313068 0.04951146  0.2082966 0.8349974
## Volume       0.135440659 0.15835970  0.8552723 0.3924004
```

```r
summary(glm.fits)$coef[,4]
```

```
## (Intercept)        Lag1        Lag2        Lag3        Lag4        Lag5
##   0.6006983   0.1452272   0.3983491   0.8243333   0.8514445   0.8349974
##      Volume
##   0.3924004
```

The predict() function can be used to predict the probability that the market will go up, given values of the predictors. The type = "response" option tells R to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the logit. If no data set is supplied to the predict() function, then the probabilities are computed for the training data that was used to fit the logistic regression model. Here we print only the first ten probabilities. We know that these values correspond to the probability of the market going up, rather than down, because the contrasts() function indicates that R has created a dummy variable with a 1 for Up.

```r
glm.probs <- predict(glm.fits, type = "response")
glm.probs[1:10]
```

```
##         1         2         3         4         5         6         7 
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509 
##         8         9        10 
## 0.5092292 0.5176135 0.4888378
```

```r
contrasts(Direction)
```

```
##      Up
## Down  0
## Up    1
```

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, Up or Down. The following two commands create a vector of class predictions based on whether the predicted probability of a market increase is greater than or less than 0.5.

```r
glm.pred <- ifelse(glm.probs > .5, "Up", "Down")
```

The function ifelse creates a vector of 1,250 elements with Down if predicted probability of a market increase exceeds 0.5, and Up if it is smaller than 0.5.

In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the held out data. This will yield a more realistic classification result, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

To implement this strategy, we will first create a vector corresponding to the observations from 2001 through 2004. We will then use this vector to create a held out data set of observations from 2005.

```r
train = (Year < 2005)
Smarket.2005 <- Smarket[!train, ]
dim(Smarket.2005)
```

```
## [1] 252   9
```

```r
Direction.2005 = Direction[!train]
```

The object train is a vector of $1,250$ elements, corresponding to the observations in our data set. The elements of the vector that correspond to observations that occurred before 2005 are set to TRUE, whereas those that correspond to observations in 2005 are set to FALSE. The object train is a Boolean vector, since its elements are TRUE and FALSE. Boolean vectors can be used to obtain a subset of the rows or columns of a matrix. For instance, the command Smarket[train,] would pick out a submatrix of the stock market data set, corresponding only to the dates before 2005, since those are the ones for which the elements of train are TRUE. The ! symbol can be used to reverse all of the elements of a Boolean vector. That is, !train is a vector similar to train, except that the elements that are TRUE in train get swapped to FALSE in !train, and the elements that are FALSE in train get swapped to TRUE in !train. Therefore, Smarket[!train,] yields a submatrix of the stock market data containing only the observations for which train is FALSE — that is, the observations with dates in 2005. The output above indicates that there are 252 such observations.

We now fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the subset argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

```r
glm.fits <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                data = Smarket, family = binomial, subset = train)
```

```
glm.probs <- predict(glm.fits, Smarket.2005, type = "response")
```

Notice that we have trained and tested our model on two completely separate data sets: training was performed using only the dates before 2005, and testing was performed using only the dates in 2005. Finally, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

```
glm.pred <- ifelse(glm.probs > .5, "Up", "Down")
```

3 kNN

We can perform kNN using the `knn()` function in the `class` package. Install the package and use it in a new workspace. After installing the package this time, you just need to run `library(class)` next time to use the functions in the package. There is no need to install it again. Now we can use the `knn()` function. Use `?knn` to see the help information of the `knn()` function. Read the help information carefully to understand what are the input options and outputs.

```
?knn
```

Here is the example from the help of `knn()`:

```
# get training and test set
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
# get class factor
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
# classification using knn, with k=3
set.seed(47)
knn_pred=knn(train, test, cl, k = 3, prob=TRUE)
# see the attributes of the knn model
attributes(knn_pred)
```

```
## $levels
## [1] "c" "s" "v"
##
## $class
## [1] "factor"
##
## $prob
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [22] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
## [29] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667 1.0000000
## [36] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [43] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [50] 1.0000000 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 1.0000000
## [57] 1.0000000 1.0000000 0.5000000 1.0000000 1.0000000 1.0000000 1.0000000
## [64] 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [71] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667
```

Exercise: Use `?iris3` to see the structure of `iris3` and think about why we can get the training and test sets using the corresponding commands.

We have to use `set.seed` before `knn()` to make the results reproducible, because if several observations are tied as nearest neighbours, then `knn()` will randomly break the tie. Change seed to see if there's change in the prediction.

Make sure you understand the outputs from `knn()`. What happens if we change `prob=FALSE`? Play with the `knn()` function by changing the inputs, e.g. `k=1` etc.

Note that if you run the following code after the above one, the original `knn_pred` (with `k=3` and `prob=TRUE`) will be covered by the new `knn_pred` (with `k=5` and `prob=FALSE`).

```
knn_pred=knn(train, test, cl, k = 5, prob=FALSE)
knn_pred
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c v c
## [36] c c c c c c c c c c c c c c c v c c v v v v v c v v v v c v v v v v v
## [71] v v v v v
## Levels: c s v
```

```
attributes(knn_pred)
```

```
## $levels
## [1] "c" "s" "v"
##
## $class
## [1] "factor"
```

To get two `knn` predictions from two models, you can simply change the name of the prediction:

```
knn_pred2=knn(train, test, cl, k = 5, prob=FALSE)
```

Play with the above codes, to see how change of training/test data, change of variables or change of k will affect the prediction.

Exercise: Randomly sample 25 observations from each class of the iris data to form the training set and the rest as the test set. Obtain the predictions of the test set of `knn` with `k=5` on this training/test split. Make sure your result is reproducible.

`knn1()` in the `class` package provides an easy way to do 1NN:

```
set.seed(47)
knn1_pred=knn1(train, test, cl)
```

If we set `prob=TRUE`, `knn()` returns the probabilities associated with the predicted class, i.e. only the largest probabilities are returned. If we want to see all probabilities, we can use the `knn3Train()` function in the `caret` package.

```
#If 'caret' package is already installed in the beginning, otherwise
#install.packages("caret")
#library(caret)
set.seed(47)
knn3_pred=knn3Train(train, test, cl, k = 3, prob=TRUE)
attributes(knn3_pred)
```

```
## $prob
##               c s         v
##  [1,] 0.0000000 1 0.0000000
##  [2,] 0.0000000 1 0.0000000
##  [3,] 0.0000000 1 0.0000000
##  [4,] 0.0000000 1 0.0000000
##  [5,] 0.0000000 1 0.0000000
##  [6,] 0.0000000 1 0.0000000
```

```
##  [7,] 0.0000000 1 0.0000000
##  [8,] 0.0000000 1 0.0000000
##  [9,] 0.0000000 1 0.0000000
## [10,] 0.0000000 1 0.0000000
## [11,] 0.0000000 1 0.0000000
## [12,] 0.0000000 1 0.0000000
## [13,] 0.0000000 1 0.0000000
## [14,] 0.0000000 1 0.0000000
## [15,] 0.0000000 1 0.0000000
## [16,] 0.0000000 1 0.0000000
## [17,] 0.0000000 1 0.0000000
## [18,] 0.0000000 1 0.0000000
## [19,] 0.0000000 1 0.0000000
## [20,] 0.0000000 1 0.0000000
## [21,] 0.0000000 1 0.0000000
## [22,] 0.0000000 1 0.0000000
## [23,] 0.0000000 1 0.0000000
## [24,] 0.0000000 1 0.0000000
## [25,] 0.0000000 1 0.0000000
## [26,] 1.0000000 0 0.0000000
## [27,] 1.0000000 0 0.0000000
## [28,] 0.3333333 0 0.6666667
## [29,] 1.0000000 0 0.0000000
## [30,] 1.0000000 0 0.0000000
## [31,] 1.0000000 0 0.0000000
## [32,] 1.0000000 0 0.0000000
## [33,] 1.0000000 0 0.0000000
## [34,] 0.3333333 0 0.6666667
## [35,] 1.0000000 0 0.0000000
## [36,] 1.0000000 0 0.0000000
## [37,] 1.0000000 0 0.0000000
## [38,] 1.0000000 0 0.0000000
## [39,] 1.0000000 0 0.0000000
## [40,] 1.0000000 0 0.0000000
## [41,] 1.0000000 0 0.0000000
## [42,] 1.0000000 0 0.0000000
## [43,] 1.0000000 0 0.0000000
## [44,] 1.0000000 0 0.0000000
## [45,] 1.0000000 0 0.0000000
## [46,] 1.0000000 0 0.0000000
## [47,] 1.0000000 0 0.0000000
## [48,] 1.0000000 0 0.0000000
## [49,] 1.0000000 0 0.0000000
## [50,] 1.0000000 0 0.0000000
## [51,] 0.0000000 0 1.0000000
## [52,] 0.6666667 0 0.3333333
## [53,] 0.7500000 0 0.2500000
## [54,] 0.0000000 0 1.0000000
## [55,] 0.0000000 0 1.0000000
## [56,] 0.0000000 0 1.0000000
## [57,] 0.0000000 0 1.0000000
## [58,] 0.0000000 0 1.0000000
## [59,] 0.5000000 0 0.5000000
## [60,] 0.0000000 0 1.0000000
```

```
## [61,] 0.0000000 0 1.0000000
## [62,] 0.0000000 0 1.0000000
## [63,] 0.0000000 0 1.0000000
## [64,] 0.6666667 0 0.3333333
## [65,] 0.0000000 0 1.0000000
## [66,] 0.0000000 0 1.0000000
## [67,] 0.0000000 0 1.0000000
## [68,] 0.0000000 0 1.0000000
## [69,] 0.0000000 0 1.0000000
## [70,] 0.0000000 0 1.0000000
## [71,] 0.0000000 0 1.0000000
## [72,] 0.3333333 0 0.6666667
## [73,] 0.0000000 0 1.0000000
## [74,] 0.0000000 0 1.0000000
## [75,] 0.3333333 0 0.6666667
```

4 Standardise data

When the scales of the variables are different, we have to scale the data before applying kNN. This is to make sure that the Euclidean distance is not dominated by the variables with large scales. To scale the dataset, we can use the `scale` function. Type `?scale` to see the help of this function.

```
?scale
```

Now suppose we change the measurement of `Sepal.Width` to milimeters and those of other variables to meters.

```
# Suppose we change Sepal.width to mm while others to m
iris_c=iris[,1:4]
iris_c[,2]=iris[,2]*10
iris_c[,-2]=iris[,c(1,3,4)]/10
```

Use `summary` to see the different scales of variables

```
summary(iris_c)
```

```
##   Sepal.Length     Sepal.Width     Petal.Length     Petal.Width
##  Min.   :0.4300   Min.   :20.00   Min.   :0.1000   Min.   :0.0100
##  1st Qu.:0.5100   1st Qu.:28.00   1st Qu.:0.1600   1st Qu.:0.0300
##  Median :0.5800   Median :30.00   Median :0.4350   Median :0.1300
##  Mean   :0.5843   Mean   :30.57   Mean   :0.3758   Mean   :0.1199
##  3rd Qu.:0.6400   3rd Qu.:33.00   3rd Qu.:0.5100   3rd Qu.:0.1800
##  Max.   :0.7900   Max.   :44.00   Max.   :0.6900   Max.   :0.2500
```

Now we apply 5NN:

```
# get indexes for training data
n=25 # the number of training data in each class
NN=dim(iris)[1]/3# the total number of observations in each class
set.seed(983)
index_s=sample(which(iris$Species=="setosa"),n)
index_c=sample(which(iris$Species=="versicolor"),n)
index_v=sample(which(iris$Species=="virginica"),n)
# get training and test set
train_rand_c = rbind(iris_c[index_s,], iris_c[index_c,], iris_c[index_v,])
test_rand_c = rbind(iris_c[-c(index_s,index_c,index_v),])
```

```r
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# classification using knn, with k=5
kk=5 # number of nearest neighbours
set.seed(275)
knn_pred=knn(train=train_rand_c,test=test_rand_c,cl=train_label, k=kk, prob=FALSE)
knn_pred
```

```
##  [1] c s s s s s s s s s s s s s c s s s s s c s s c v c c c c c c v c c
## [36] c v c c c v c c c c c c c c c v v v v c v v c v s v c v v s v s c v c
## [71] v v v s c
## Levels: c s v
```

Exercise: Have a look at the predictions and compare them with the ground truth, what do you find?

Now we standardise the data first using `scale( )`:

```r
iris_s=scale(iris_c)
```

Check the standard deviation and mean of the variables are all 1 and 0, respectively.

Then if we apply 5NN using the same training and test indexes:

```r
# get training and test set
train_rand_s = rbind(iris_s[index_s,], iris_s[index_c,], iris_s[index_v,])
test_rand_s = rbind(iris_s[-c(index_s,index_c,index_v),])
# get class factor for training data
train_label= factor(c(rep("s",n), rep("c",n), rep("v",n)))
# get class factor for test data
test_label_true=factor(c(rep("s",NN-n), rep("c",NN-n), rep("v",NN-n)))
# classification using knn, with k=5
kk=5 # number of nearest neighbours
set.seed(275)
knn_pred=knn(train=train_rand_s,test=test_rand_s,cl=train_label, k=kk, prob=FALSE)
knn_pred
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c c c c c c c c v
## [36] c c v c c c c c c c c c c c v v v v c v v v v v v v v v v c v c v v
## [71] v v v v c
## Levels: c s v
```

Exercise: Compare the predictions with those without scaling. What do you find?