DSA1101 Tutorial 6 Working File

Dawn Cheung

2024-04-28

Q1: KNN and N-fold Cross Validation

Loan managers often need to take into account an applicant’s demographic and socio-economic profiles in deciding whether to approve a loan to the applicant, to minimize losses due to defaults. In this exercise we will build and evaluate a classifier based on the German Credit Data to predict whether an applicant is considered as having good or bad credit risk. The features or predictors include (1) loan duration (in months), (2) credit amount, (3) Installment rate in percentage of disposable income and (4) age in years

a. Read and explore the data from the file German_credit.csv.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.2
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
come errors
```

```
library(dplyr)
library(class)
```

```
## Warning: package 'class' was built under R version 4.3.2
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.3.2
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.3.2
```

```
data = read.csv("~/Github/DSA1101 Slayers/datasets/German_credit.csv") #Q1
attach(data)
glimpse(data)
```

```
## Rows: 1,000
## Columns: 5
## $ Creditability <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…
## $ Duration      <int> 18, 9, 12, 12, 12, 10, 8, 6, 18, 24, 11, 30, 6, 48, 18, …
## $ Amount        <int> 1049, 2799, 841, 2122, 2171, 2241, 3398, 1361, 1098, 375…
## $ Instalment    <int> 4, 2, 2, 3, 4, 1, 1, 2, 4, 1, 2, 1, 1, 2, 2, 2, 1, 1, 4,…
## $ Age           <int> 21, 36, 23, 39, 38, 48, 39, 40, 65, 23, 36, 24, 31, 31, …
```

b. Standardize the input features.

```
stand.X = scale(data[,2:5])
#jbfhdkjsvlkdjh
#end me

#alt method:
#credit[,2:5] = lapply(credit[,2:5], scale)
#lapply applies the function to the input???
#apply scale function to credit[,2:5]
#mostly for convienience when applying 1 function to multiple columns
#apply() can allow u to configure more (like combine diff cols?) but eh lapply does the job
```

c. Randomly select 800 customer records to form the training data, and the remaining 200 records will be the test data.

```
setA = sample(1:1000, size = 800)
q = 1:1000
q = q[which(q != setA)]
```

```
## Warning in q != setA: longer object length is not a multiple of shorter object
## length
```

```
setB = sample(q, size = 200)

#SIMPLER WAY TO GET SETB
#test.data = credit[-setA,]

set.seed(897)

train.X = stand.X[setA, ]
test.X = stand.X[setB, ]
train.Y = data[setA, 1]
test.Y = data[setB, 1]
```

d. Use 1-nearest neighbor classifier for the training data to predict if a loan applicant is credible for the 200 test points. Compute the accuracy of the classifier.

```
knn.pred = knn(train.X, test.X, train.Y, k = 1)
knn.pred
```

```
##   [1] 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1
##  [75] 0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0
## [112] 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1
## [149] 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1
## [186] 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1
## Levels: 0 1
```

```
killme = table(knn.pred, test.Y)
acc = (killme[1,1] + killme[2,2]) / 200
acc
```

```
## [1] 0.94
```

e. Use N-folds cross validation with N = 5 to find the average accuracy for the 1-nearest neighbor classifier.

```
setA = sample(1:1000, size = 200)
q = 1:1000
#i know this can be automated but im too tired for this
q = q[which(q != setA)]
setB = sample(q, size = 200)
q = q[which(q != setB)]
```

```
## Warning in q != setB: longer object length is not a multiple of shorter object
## length
```

```
setC = sample(q, size = 200)
q = q[which(q != setC)]
```

```
## Warning in q != setC: longer object length is not a multiple of shorter object
## length
```

```
setD = sample(q, size = 200)
q = q[which(q != setC)]
```

```
## Warning in q != setC: longer object length is not a multiple of shorter object
## length
```

```
setE = sample(q, size = 200)

COMBINE = data.frame(setA, setB, setC, setD, setE)

#SIMPLER WAY:
#n_folds=5 # each fold has 200 data points
#folds_j <- sample(rep(1:n_folds, length.out = dim(credit)[1] ))
#rep stands for replication
#so rep(1:n_folds, length.out = dim(credit)[1] ) returns 1 2 3 4 5 1 2 3 4 5 1 2 3...

to_compare = c()

please <- function(HELPME = 1) {
  aveacc_count = c()
  for (i in 1:5) {
    train.X = stand.X[-unlist(COMBINE[,i]), ]
    test.X = stand.X[unlist(COMBINE[,i]), ]
    train.Y = data[-unlist(COMBINE[,i]), 1]
    test.Y = data[unlist(COMBINE[,i]), 1]

    knn.pred = knn(train.X, test.X, train.Y, k = HELPME)
    knn.pred

    killme = table(knn.pred, test.Y)
    acc = (killme[1,1] + killme[2,2]) / 200
    aveacc_count = c(aveacc_count, acc)
  }
  to_compare = c(to_compare, mean(aveacc_count))
  return(mean(aveacc_count))
  #i want to mf kms
}

please()
```

```
## [1] 0.623
```

f. Repeat question 1e for K-nearest neighbor classifiers where K = 1, 2, …100.

```
for (i in 1:100) {
  please(i)
  to_compare = c(to_compare, please(i))
}
#nvm goodnight
```

g. Compare the 100 classifiers above, which few values of K give the best average accuracy?

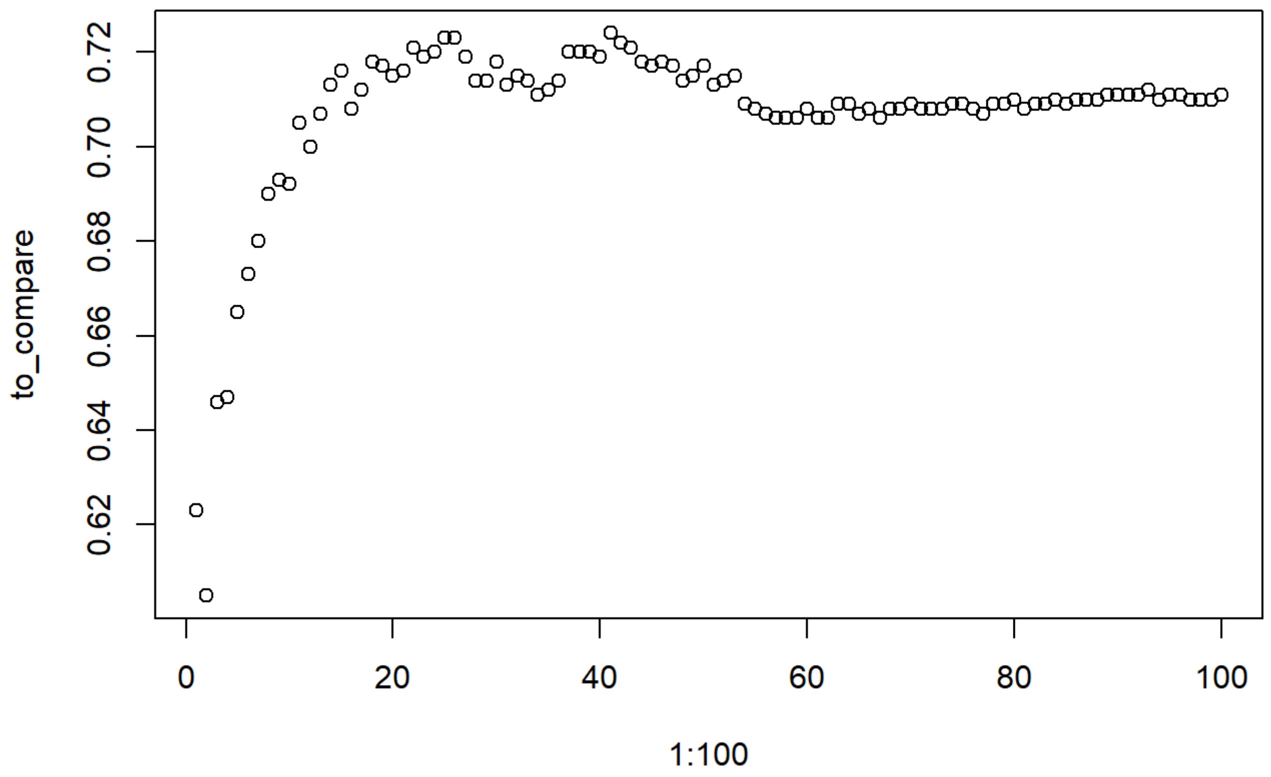
```
to_compare
```

```
##   [1] 0.623 0.605 0.646 0.647 0.665 0.673 0.680 0.690 0.693 0.692 0.705 0.700
##  [13] 0.707 0.713 0.716 0.708 0.712 0.718 0.717 0.715 0.716 0.721 0.719 0.720
##  [25] 0.723 0.723 0.719 0.714 0.714 0.718 0.713 0.715 0.714 0.711 0.712 0.714
##  [37] 0.720 0.720 0.720 0.719 0.724 0.722 0.721 0.718 0.717 0.718 0.717 0.714
##  [49] 0.715 0.717 0.713 0.714 0.715 0.709 0.708 0.707 0.706 0.706 0.706 0.708
##  [61] 0.706 0.706 0.709 0.709 0.707 0.708 0.706 0.708 0.708 0.709 0.708 0.708
##  [73] 0.708 0.709 0.709 0.708 0.707 0.709 0.709 0.710 0.708 0.709 0.709 0.710
##  [85] 0.709 0.710 0.710 0.710 0.711 0.711 0.711 0.711 0.712 0.710 0.711 0.711
##  [97] 0.710 0.710 0.710 0.711
```

```
sort(to_compare, TRUE)[1]
```

```
## [1] 0.724
```

```
plot(x = 1:100, to_compare)
```



Q2: Decision Trees

Consider the famous Iris Flower Data set which was first introduced in 1936 by the famous stastistician Ronald Fisher. This data set consists of 50 observations from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each observation: the length and the width of the sepals and petals (in cm).

a. Use decision tree method to predict Iris species based on all four features.

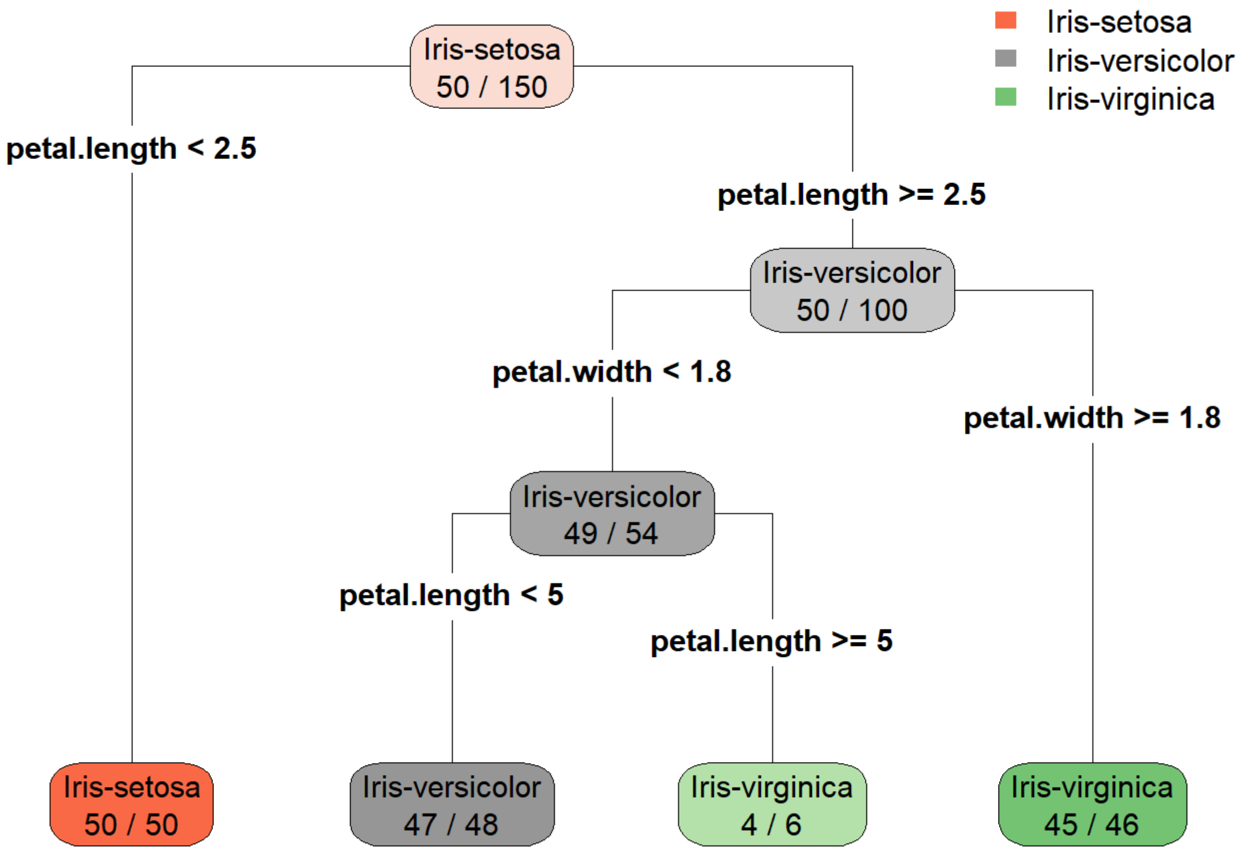
```
iris = read.csv("~/Github/DSA1101 Slayers/datasets/iris.csv")
head(iris)
```

```
##   sepal.length sepal.width petal.length petal.width       class
## 1          5.1         3.5          1.4         0.2 Iris-setosa
## 2          4.9         3.0          1.4         0.2 Iris-setosa
## 3          4.7         3.2          1.3         0.2 Iris-setosa
## 4          4.6         3.1          1.5         0.2 Iris-setosa
## 5          5.0         3.6          1.4         0.2 Iris-setosa
## 6          5.4         3.9          1.7         0.4 Iris-setosa
```

```
attach(iris)
fit <- rpart(class ~ sepal.length + sepal.width + petal.length + petal.width,
             method = "class", #tells R to return the category of response (yes or no)
             data = iris,
             control = rpart.control(minsplit = 1), #tells R how to split the node; minsplit
= 1 means a branch is created when there is at least 1 observation in that branch => tells R
how big we want the tree to be
             parms = list(split = "information")) # what criteria to use to choose the root n
ode & internal nodes; either info gained or gini index; default val is gini
```

b. Visualize the decision tree above, using the rpart.plot function.

```
rpart.plot(fit, type = 4, extra = 2, clip.right.labs = FALSE)
```



c. What are the more important features in the fitted tree above?

petal length and petal width

```
#use predict() function to do prediction lol
predict(fit, newdata = data.frame(sepal.length = 5.1, sepal.width = 3.3, petal.length = 3, pe
tal.width = 1.8), type = "class")
```

```
##              1
## Iris-virginica
## Levels: Iris-setosa Iris-versicolor Iris-virginica
```

```
#if you want the most likely type of flower, type = "class" is important here bc type = "pro
b" will give u a vector of probability
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.