ML_block2_lab

Selen Karaduman, Hong Zhang

12/16/2021

Contents

1 Data description and research objectives

In this laboratory, we chose “Adult Data Set” from UCI Machine Learning Repository. This data-set was extracted by Barry Becker from a 1994 census database. The total number of observations in this data-set is 48842 and it contains a total of 14 Attributes such as age, gender, work status and education. The URL in citation one(“Adult” 1996) contains a description of these features.

In the adult data-set, we find that there are some missing values (shown as “?” in the data-set) The number of objects containing missing values is 2366, which would have affected the performance of the model. So after reading the data, we cleaned the data-set to remove the objects that contained missing values.

In this laboratory, we will implement a classification problem and predict whether income exceeds $50K/yr based on census data. The cleaned data-set is divided into a training set, a validation set and a test set (40/30/30). We will compare the two different classification models, discuss and conclude.

2 Descriptions of models used and laboratoryal design

In this laboratory, the models we use are the decision tree model and the support vector machine model.

a. Decision trees

The packages used are “tree” package and “rpart” package.

First, a decision tree is constructed with the default parameter settings, denoted by “tree_a” in R script. The default tree model is then optimized by the accuracy rate returned by the validation data-set.

We then tried to build different tree models, setting a tree with a minimum number of nodes of 5000 (denoted by tree_b) and a tree with a default deviance of 0.0005 (denoted by tree_b), respectively, and selected the tree model to be optimal in the next step by the correct rate returned.

After that, we selected the tree model to be optimized as tree_c, and the best leaf tree was selected by comparing the Bias-variance trade off plot of the training data and the test data as 41.

Finally, the tree_c is pruned using the equation prune.tree(…) , thus giving us an optimal tree model (denoted by tree_optimal).

With this optimal tree model, the training set is classified

b. Support vector machines

The data-set was first divided into a training data-set, a validation data-set and a test data-set (40/30/30). Again, the training data set was used as the default model, “rbfdot” was used as the kernel function, and the parameter sigma was set to 0.01. The model was constructed with different parameters. The best parameters are determined by the classification errors returned from the validation data-set.

The model was validated for parameters C in the range 0.1 to 5.0, and it was found that the validation set returned the lowest classification error for parameter C of 0.3. Based on the above parameters, the final model was constructed.

3 Result

3.1 Tree model

3.1.1 Data reading and data cleaning

Read the data-set “adult.data” and factorise the response in the data-set. Since the data-set contains some missing values, the data-set needs to be cleaned to remove the objects whose features contain missing values.

3.1.2 Tree models

Different tree models are constructed: a default tree, tree with a minimum node size of 5000 and tree with a minimum deviation of 0.0005.

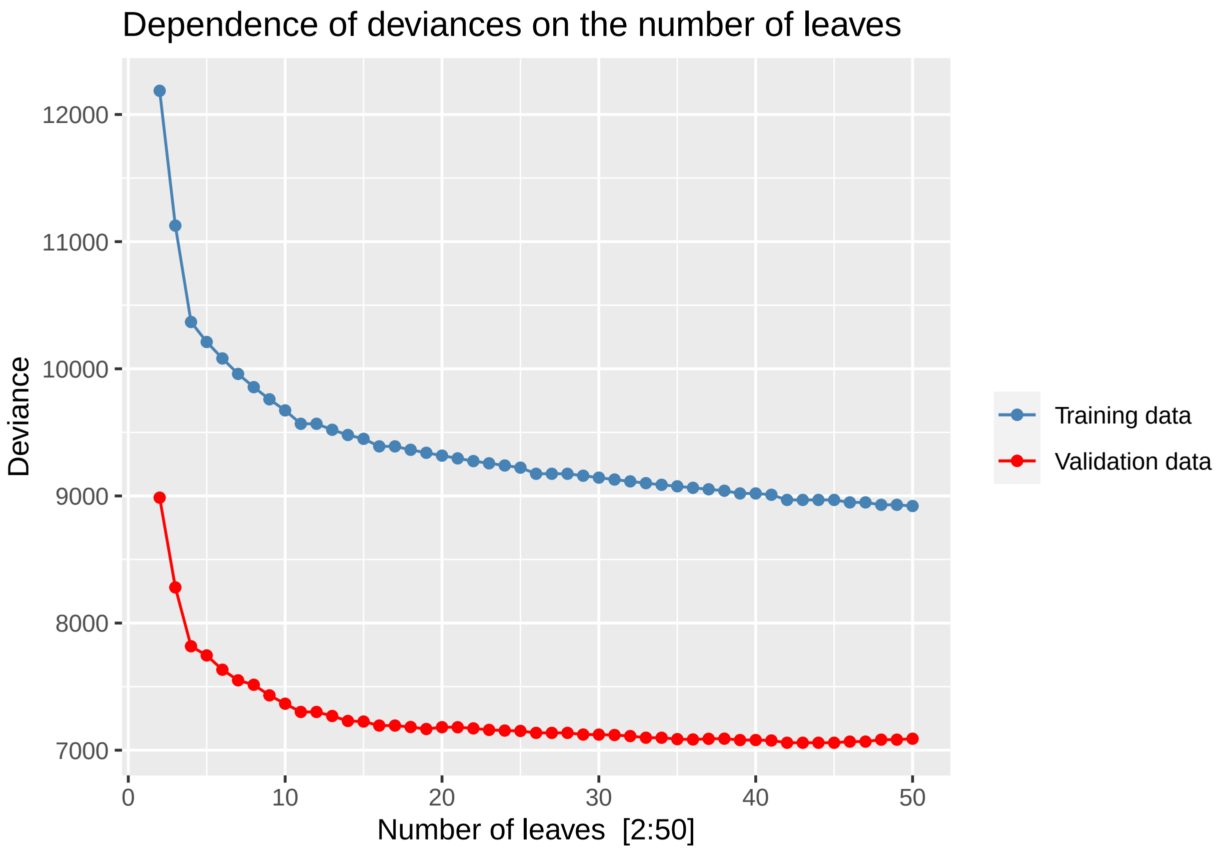
We generate a table with misclassification errors and choose tree_c (tree with a minimum deviation of 0.0005) as the target model for further optimization.

```
## [1] 0.2104217
```

```
##                           tree_a    tree_b    tree_c
## Train_error_rates:  0.2068394 0.2068394 0.1651900
## Valid_error_rates:  0.2104217 0.2104217 0.1806138
```

3.1.3 Optimal tree model

Create a Bias-variance tradeoff plot, choosing the right number of leaves according to the principle of Early stopping, the best number of leaves returned is 42.



```
## [1] 42
```

3.1.4 Prediction

The prediction accuracy of the optimization tree model is 0.8220554.

```
##          Yfit
##            <=50K  >50K
##    <=50K   6475   342
##    >50K    1270   972
```

```
## [1] 0.8220554
```

3.2 SVM method

3.2.1 Data clean

SVM methods are sensitive to missing values in the data-set, so data cleaning is required.

3.2.2 Optimal parameters

The model is constructed using the train data as the data-set, and the error returned by the prediction of the validation data-set is used to select the appropriate parameters.

The kernel function was selected as "rbfdot," the parameter sigma was set to 0.05, and the parameter c was set to a range of 0.1 to 5.0 for parameter optimization.

3.2.3 Optimal SVM

The optimal parameter c is 0.8, when the validation set returns the smallest error and the optimized SVM is constructed (denoted by svm_model). In constructing the optimal SVM, we have used a generalization error criterion that allows the model to have better stability.

```r
svm_model <- ksvm(X..50K~., data = rbind(train,valid), kernel="rbfdot",kpar=list(sigma=0.05),
                  C = 0.8, scaled=FALSE)
```

3.2.4 SVM prediction

We used the same test data-set for prediction as in the tree model to ensure consistency when comparing the two methods. The optimal SVM method returned an accuracy of 89%.

```
## [1] 0.8919307
```

4 Discussion and conclusion

In this laboratory, the tree model returned a classification accuracy of 82%; the SVM method returned a classification accuracy of 89%. Our group discussed that the SVM method had some shortcomings in the degree of optimisation. For example, we only tested a sigma value of 0.05, and the default kernel function chosen was "rbfdot"; choosing a different combination of kernels would have provided a more accurate model. However, in this experiment, the categories contained in the individual features in the selected data-set were too complex, which caused difficulties in dividing the data in the training set. Specifically, there may be a mismatch between the data and the model when the model is subsequently optimal. To address this situation, our group took the approach of making the feature categories of the training set as inclusive as possible. This ensures model matching for the subsequent validation data-set and the test data-set.

Given that the SVM model has the following problems.

(1) SVM performs very well with small samples, as its generalisation ability is superior to that of classification algorithms, but is less effective when faced with large, high-dimensional data.

(2) It is sensitive to missing data.

Among other things, sensitivity to missing data was verified in this experiment. Missing data in a data-set can easily be cleaned up at the beginning to get the "ideal" data sample. However, in the subsequent classification process, it is difficult to get the training set samples to contain all the features, and there is a mismatch between the data and the model.

In this regard, we propose the following idea: can the mismatch between the model and the data-set be solved by numericising all the features in the data-set? In the case where all features are numeric, it is easy to apply the SVM model without worrying about the mismatch. A possible problem caused by feature numeric is a further reduction in the explanatory power of the model.

In summary, we believe that tree-based models require relatively less data pre-processing and should be preferred for fast implementation of classification problems. the parameter calls and optimisation of SVM would take up extra time and computational resources.

Reference

Code Appendix

```
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
# Tree----------------------------------------------------------------------
library(tree)
library(rpart)
library(ggplot2)
# Read data-----------------------------------------------------------------
data_read = read.csv("adult.data")
data = data_read
data$X..50K = as.factor(data$X..50K)
# Clean and divide data-----------------------------------------------------
for(i in 1:nrow(data)){
  for(j in 1:ncol(data)){
    if(isTRUE(as.character(data[i,j]) == " ?")){
      data = data[-i,]
    }
  }
}
# divide data
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = data[id, ]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.3))
valid = data[id2, ]
id3 = setdiff(id1, id2)
test = data[id3, ]
```

```r
# Default tree--------------------------------------------------------------------
tree_a = tree(X..50K~., train)
#--train data
Yfit = predict(tree_a, train, type="class")
train_error1 = 1 - sum(diag(table(train$X..50K, Yfit))) / sum(table(train$X..50K,Yfit))
#--valid data
Yfit = predict(tree_a, valid, type="class")
valid_error1 = 1 - sum(diag(table(valid$X..50K,Yfit))) / sum(table(valid$X..50K,Yfit))
valid_error1 # 0.2104217

# Smallest node size 5000-------------------------------------------------------
tree_b = tree(X..50K~., train, minsize = 5000)
#--train data
Yfit = predict(tree_b, train, type="class")
train_error2 = 1 - sum(diag(table(train$X..50K,Yfit))) / sum(table(train$X..50K,Yfit))
#--valid data
Yfit = predict(tree_b, valid, type="class")
valid_error2 = 1 - sum(diag(table(valid$X..50K,Yfit))) / sum(table(valid$X..50K,Yfit))

# Minimum deviance 0.0005-------------------------------------------------------
tree_c = tree(X..50K~., train, mindev = 0.0005)
#--train data
Yfit = predict(tree_c, train, type="class")
train_error3 = 1 - sum(diag(table(train$X..50K,Yfit))) / sum(table(train$X..50K,Yfit))
#--valid data
Yfit = predict(tree_c, valid, type="class")
valid_error3 = 1 - sum(diag(table(valid$X..50K,Yfit))) / sum(table(valid$X..50K,Yfit))
# 0.1806138

# Misclassification rates
data.frame(tree_a=c(train_error1,valid_error1),
           tree_b=c(train_error2,valid_error2),
           tree_c=c(train_error3,valid_error3),
           row.names = c("Train_error_rates: ","Valid_error_rates: "))

# Optimal tree--------------------------------------------------------------------
# Choose the optimal tree depth
trainScore = rep(0,50)
validScore = rep(0,50)

for(i in 2:50){
  prunedTree = prune.tree(tree_c, best=i)
  pred = predict(prunedTree, newdata=valid, type="tree")
  trainScore[i] = deviance(prunedTree)
  validScore[i] = deviance(pred)
}

df = data.frame(trainScore[2:50],validScore[2:50])
ggplot(df)+
  geom_point(aes(x=2:50, y=validScore[2:50],color = "Validation data"))+
  geom_point(aes(x=2:50, y=trainScore[2:50],color = "Training data"))+
  geom_line(aes(x=2:50, y=validScore[2:50],color = "Validation data"))+
  geom_line(aes(x=2:50, y=trainScore[2:50],color = "Training data"))+
  xlab("Number of leaves  [2:50]") + ylab("Deviance")+
  ggtitle("Dependence of deviances on the number of leaves")+
  scale_colour_manual("",
                      breaks = c("Training data","Validation data"),
                      values = c("steelblue","red"))+
  theme(plot.title = element_text(hjust = 0.5))+
  theme_set(theme_bw())

# Optimal amount of leaves: 42
which.min(validScore[2:50]) + 1

# Variable importance
optimal_tree = prune.tree(tree_c, best=42)


# prediction------------------------------------------------------------------
# Optimal tree
tree_optimal = prune.tree(tree_c, best=42)
Yfit = predict(tree_optimal, test, type="class")
#--Confusion matrix
confusion_optimal = table(test$X..50K, Yfit)
confusion_optimal
#--Accuracy rate
accuracy_optimal = sum(diag(confusion_optimal)) / sum(confusion_optimal)
accuracy_optimal # 0.8220554

# Lab block2--------------------------------------------------------------------
library(kernlab)

# Read data-------------------------------------------------------------------
data_read = read.csv("adult.data")
data = data_read
data$X..50K = as.factor(data$X..50K)
# Clean data
for(i in 1:nrow(data)){
  for(j in 1:ncol(data)){
    if(isTRUE(as.character(data[i,j]) == " ?")){
      data = data[-i,]
    }
  }
}
# Divide data
n <- sample(nrow(data))
data <- data[n,]
valid <- data[1:9058, ] # training data-set
train <- data[9059:18116, ] # validation data-set
trva <- data[1:18117, ] # training combine validation data-set
test <- data[18119:30194, ]  # test data-set

# SVM-------------------------------------------------------------------------
# choose C
by = 0.1
err_va = NULL
```

```r
for(i in seq(by,5,by)){ # set C parameters from 0.1 to 5.0
  filter = ksvm(X..50K~.,data=train,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype = predict(filter,valid[,-15])
  t = table(mailtype,valid[,15])
  err_va = c(err_va,(t[1,2]+t[2,1])/sum(t))
}


# Svm model
c = which.min(err_va)*by # 0.8

svm_model <- ksvm(X..50K~., data = rbind(train,valid), kernel="rbfdot",kpar=list(sigma=0.05),
                  C = 0.8, scaled=FALSE)

# Reset test data
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
train = data[id, ]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.3))
valid = data[id2, ]
id3 = setdiff(id1, id2)
test = data[id3, ]

pre <- predict(svm_model, test[,-15]) # test error
t <- table(pre, test[,15])
error <- (t[1,2]+t[2,1])/sum(t)
1 - error
```

"Adult." 1996. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Adult.