SMM636 Machine Learning (PRD2 A 2019/20)

R exercises 5: Fisher‘s linear discriminant analysis

DR. Rui Zhu

2020-01-21

In R exercise 5, you will know

- How to apply LDA using the `lda()` function
- How to use LDA in the `caret` package
- How to tune the number of dimensions of the `lda2()` function in the `caret` package
- Create an interactive app that can show different projections of a dataset
- How to use LDA as a dimension reduction/distance metric learning method
- More real data examples

Don’t forget to change your working directory!

1 The lda function

The `lda()` function is in the library `MASS`. To use `lda()`,

```
install.packages(MASS)
library(MASS)
```

Prepare training and test set:

```
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 = rbind(iris[index_s,], iris[index_c,], iris[index_v,])
test_rand = rbind(iris[-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)))
```

Use `?lda` to see the help information of `lda()`.

```
?lda
```

To fit the lda model:

```
fit1=lda(Species~.,data=train_rand)
fit2=lda(train_rand[,-5],train_rand[,5])
```

The two fits have the same results: you can use either way.

```
fit1
```

```
## Call:
## lda(Species ~ ., data = train_rand)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            4.916       3.364        1.456       0.228
## versicolor        5.824       2.788        4.208       1.320
## virginica         6.628       2.936        5.568       2.056
##
## Coefficients of linear discriminants:
##                    LD1       LD2
## Sepal.Length  1.117402 -1.208336
## Sepal.Width   1.572141 -1.223234
## Petal.Length -2.274850  1.692932
## Petal.Width  -2.950060 -3.105155
##
## Proportion of trace:
##    LD1    LD2
## 0.9927 0.0073
```

The prior probabilities are estimated as the class proportions for the training set. Group means are the group means in the original feature spaces. Coefficients of linear discriminants are the two directions we find using LDA. Proportions of traces are related to the singular values obtained for the two directions when solving LDA.

Now we can predict the labels for the test set:

```
pred=predict(fit1,test_rand[,-5])
pred
```

```
## $class
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     setosa     setosa     setosa     setosa
## [19] setosa     setosa     setosa     setosa     setosa     setosa
## [25] setosa     versicolor versicolor versicolor versicolor versicolor
## [31] versicolor versicolor versicolor versicolor versicolor versicolor
## [37] virginica  versicolor versicolor versicolor versicolor versicolor
## [43] versicolor versicolor versicolor versicolor versicolor versicolor
## [49] versicolor versicolor virginica  virginica  virginica  virginica
## [55] virginica  virginica  virginica  virginica  virginica  virginica
## [61] virginica  virginica  virginica  virginica  virginica  versicolor
## [67] virginica  virginica  virginica  virginica  virginica  virginica
## [73] virginica  virginica  virginica
## Levels: setosa versicolor virginica
##
## $posterior
##          setosa   versicolor    virginica
## 2   1.000000e+00 3.008379e-18 3.538314e-38
## 3   1.000000e+00 1.308059e-19 5.015881e-40
## 4   1.000000e+00 1.925777e-16 7.605686e-36
## 5   1.000000e+00 7.223914e-23 2.861718e-44
## 7   1.000000e+00 2.077910e-18 4.038444e-38
## 8   1.000000e+00 1.667473e-20 3.878861e-41
## 11  1.000000e+00 1.443816e-24 1.978199e-46
## 15  1.000000e+00 1.336212e-31 1.142770e-55
## 16  1.000000e+00 1.040991e-28 3.492776e-51
## 17  1.000000e+00 9.362676e-26 2.398786e-47
## 19  1.000000e+00 1.081867e-23 7.220463e-45
## 20  1.000000e+00 1.143381e-22 1.199127e-43
## 21  1.000000e+00 1.583518e-20 4.487786e-41
## 22  1.000000e+00 6.720804e-21 5.659832e-41
## 24  1.000000e+00 4.332741e-15 6.060437e-33
## 26  1.000000e+00 9.083816e-17 3.447996e-36
## 27  1.000000e+00 2.387590e-17 2.723251e-36
## 28  1.000000e+00 3.419922e-22 2.492776e-43
## 29  1.000000e+00 1.698692e-22 9.767506e-44
## 30  1.000000e+00 1.223818e-16 4.428166e-36
## 34  1.000000e+00 1.722068e-29 6.269034e-53
## 39  1.000000e+00 9.815721e-17 2.770606e-36
## 40  1.000000e+00 5.263423e-21 8.848915e-42
## 41  1.000000e+00 4.341499e-22 6.535408e-43
## 46  1.000000e+00 1.153166e-16 9.039744e-36
## 51  5.444586e-17 9.999619e-01 3.811277e-05
## 54  1.150915e-22 9.994551e-01 5.448824e-04
## 55  2.467753e-22 9.966092e-01 3.390796e-03
## 59  5.578775e-19 9.999364e-01 6.356835e-05
## 62  5.287155e-20 9.995693e-01 4.306780e-04
## 63  1.339914e-17 9.999984e-01 1.603325e-06
## 64  1.472511e-23 9.967291e-01 3.270915e-03
## 66  3.297843e-16 9.999822e-01 1.783585e-05
## 68  3.693524e-16 9.999995e-01 5.038986e-07
## 69  5.308454e-27 9.161557e-01 8.384434e-02
## 70  1.232802e-17 9.999971e-01 2.891812e-06
## 71  3.165850e-28 3.932273e-01 6.067727e-01
## 73  1.790259e-28 7.783853e-01 2.216147e-01
## 75  5.200790e-17 9.999871e-01 1.289802e-05
## 77  9.946587e-22 9.988108e-01 1.189243e-03
## 78  3.378476e-26 7.716676e-01 2.283324e-01
## 80  2.100720e-11 1.000000e+00 1.305340e-08
## 81  7.834371e-18 9.999968e-01 3.156425e-06
## 83  2.413397e-16 9.999974e-01 2.618411e-06
## 85  1.735815e-25 9.797986e-01 2.020140e-02
## 89  2.381925e-18 9.999767e-01 2.327650e-05
## 90  2.717177e-21 9.998038e-01 1.961518e-04
## 93  5.079268e-18 9.999908e-01 9.223483e-06
## 97  1.588102e-19 9.999407e-01 5.925685e-05
## 98  5.181828e-18 9.999753e-01 2.469249e-05
## 101 1.124054e-52 8.299593e-09 1.000000e+00
## 103 6.879950e-42 2.738271e-05 9.999726e-01
## 104 1.088041e-38 1.288032e-03 9.987120e-01
## 105 3.439207e-46 1.711980e-06 9.999983e-01
## 107 4.348194e-35 3.102706e-02 9.689729e-01
## 108 1.771023e-41 1.762553e-04 9.998237e-01
## 109 2.189008e-42 1.370689e-04 9.998629e-01
## 114 8.191672e-42 9.132760e-05 9.999087e-01
## 117 3.473420e-35 8.622490e-03 9.913775e-01
## 118 4.653037e-43 5.982272e-06 9.999940e-01
## 123 1.302879e-48 8.377448e-07 9.999992e-01
## 124 2.460975e-31 8.030991e-02 9.196901e-01
## 126 1.884210e-35 5.538134e-03 9.944619e-01
## 127 8.358994e-30 1.818787e-01 8.181213e-01
## 132 2.674531e-34 2.501318e-03 9.974987e-01
## 134 2.162995e-28 7.846500e-01 2.153500e-01
## 137 9.680411e-45 1.328863e-06 9.999987e-01
## 139 2.636554e-29 2.439622e-01 7.560378e-01
## 142 2.840735e-35 4.083885e-04 9.995916e-01
## 143 6.636168e-39 7.998850e-04 9.992001e-01
## 144 1.577060e-45 1.238014e-06 9.999988e-01
## 146 8.836266e-39 6.057836e-05 9.999394e-01
## 148 9.073406e-35 3.524469e-03 9.964755e-01
## 149 5.497957e-41 2.065776e-05 9.999793e-01
## 150 8.794256e-34 2.411933e-02 9.758807e-01
##
## $x
##            LD1         LD2
## 2    7.2463845  0.25155588
## 3    7.5648174  0.07928287
## 4    6.8408931  0.66102633
## 5    8.3014094 -0.60321830
## 7    7.2450145 -0.18575268
## 8    7.7594961 -0.18927819
## 11   8.6780991 -1.03958271
## 15  10.2791572 -2.39776695
## 16   9.5238063 -2.87937835
## 17   8.8574853 -2.24384698
## 19   8.4205575 -1.49633579
## 20   8.2050867 -1.10992095
## 21   7.7514866 -0.33402592
## 22   7.7528665 -1.29811297
## 24   6.3740339 -0.78074820
## 26   6.9031545  0.46930880
## 27   6.9419990 -0.64101587
## 28   8.1401905 -0.55326873
## 29   8.2104614 -0.60023853
## 30   6.8823623  0.58716258
## 34   9.8033949 -1.94132669
## 39   6.9151687  0.68643040
## 40   7.8712362 -0.31011174
## 41   8.0766742 -0.96070356
## 46   6.8396383  0.06187397
## 51  -1.1397227 -0.67010422
## 54  -2.3433508  1.36877279
## 55  -2.3948010 -0.05645141
## 59  -1.5358346  0.32142252
## 62  -1.8408735 -0.25326992
## 63  -1.0568461  1.81847485
## 64  -2.6170265  0.78436806
## 66  -0.9497022 -0.69315984
## 68  -0.7217408  1.61781800
## 69  -3.4458212  0.87069664
## 70  -1.0996853  1.45503004
## 71  -3.7763736 -0.41370374
## 73  -3.7723788  1.06006573
## 75  -1.0768598  0.05520991
## 77  -2.2195446  0.23014987
## 78  -3.3568446 -0.48662335
## 80   0.3742151  0.84521556
## 81  -1.1411546  1.52889379
## 83  -0.8567828  0.65820060
## 85  -3.0820294  0.85877755
## 89  -1.3585968  0.56096841
## 90  -2.0289226  1.12412592
## 93  -1.2414820  0.94981728
## 97  -1.6315559  0.73175154
## 98  -1.3003401  0.29687701
## 101 -7.9670615 -1.16145074
## 103 -6.1372735 -0.68838024
## 104 -5.6209356  0.82427827
## 105 -6.8752354 -0.44318763
## 107 -5.0168128  1.45353159
## 108 -6.0959294  0.80099542
## 109 -6.2578016  1.16882429
## 114 -6.1453349  0.40178299
## 117 -5.0127562  0.29099449
## 118 -6.3240059 -1.34813861
## 123 -7.3061351  0.49612669
## 124 -4.3429686 -0.11612752
## 126 -5.0535720  0.04697894
## 127 -4.0700096 -0.28691065
## 132 -4.8280584 -1.47665452
## 134 -3.7557065  1.03168189
## 137 -6.6049012 -1.65043167
## 139 -3.9790617 -0.28989042
## 142 -4.9736715 -2.54441340
## 143 -5.6516455  0.51611126
## 144 -6.7480778 -1.19155738
## 146 -5.5818509 -2.01112962
## 148 -4.9203131 -0.83791614
## 149 -5.9666653 -1.55766914
## 150 -4.7732569  0.33882285
```

`pred` has three parts: `class`, `posterior` and `x`. `class` is the predicted labels. `posterior` gives you posterior probabilities of belonging to one class for each instance. `x` is the projected features of the test set.

To calculate the classification accurary:

```
acc = mean(test_rand[,5]==pred$class)
acc
```

```
## [1] 0.9733333
```

Exercise: Draw a scatter plot of the projected training data using LDA.

2 LDA in the caret package

We can also use the `lda` function from the `caret` package.

```
library(caret)
ldaFit=train(train_rand[,-5],train_rand[,5],method="lda",
                 trControl=trainControl(method = "none"))
ldaFit$finalModel
```

```
## Call:
## lda(x, y)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            4.916       3.364        1.456       0.228
## versicolor        5.824       2.788        4.208       1.320
## virginica         6.628       2.936        5.568       2.056
##
## Coefficients of linear discriminants:
##                    LD1       LD2
## Sepal.Length  1.117402 -1.208336
## Sepal.Width   1.572141 -1.223234
## Petal.Length -2.274850  1.692932
## Petal.Width  -2.950060 -3.105155
##
## Proportion of trace:
##    LD1    LD2
## 0.9927 0.0073
```

```
pred2=predict(ldaFit,test_rand[,-5])
acc2=mean(pred2==test_rand[,5])
acc2
```

```
## [1] 0.9733333
```

Check the final model to see if it's the same as the previous section.

The `lda` function uses all $C-1$ directions for classification, so there's no parameter to be tuned in the training phase. However, we don't have to use all $C-1$ directions. The `lda2` function in the `caret` package allows us to tune this parameter, i.e. the number of directions to use for classification.

```
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated ten times
  repeats = 5)
set.seed(836)
lda2Fit=train(train_rand[,-5],train_rand[,5],method="lda2",
                    trControl=fitControl)
pred3=predict(lda2Fit,test_rand[,-5])
acc3=mean(pred3==test_rand[,5])
acc3
```

```
## [1] 0.9866667
```

The increase in classification accuracy shows that using only one direction rather than two directions can result in better classification performance for this training/test split.

Exercise: Repeat the random split procedure 50 times. For each training/test split, tune the number of directions based on 10-fold cross-validation. Get a boxplot of the classification accuracies. Store the tuned number of directions for each split in a vector.

3 Create an interactive app that can show different projections of a dataset

```r
library(shiny)
library(shinythemes)
ui <- fluidPage(theme = shinytheme("lumen"),
                titlePanel("Projection of iris data"),
                sidebarLayout(
                  sidebarPanel(
                    # Select type of projection to plot
                    selectInput(inputId = "type", label = strong("Projection type"),
                                choices = c("PCA","LDA"),
                                selected = "PCA")
                    ),
                  # Output: Description, lineplot, and reference
                  mainPanel(
                    plotOutput(outputId = "projection")
                  )
                )
)

# Define server logic required to draw a projection plot ----
server <- function(input, output) {
  output$projection <- renderPlot({
    data(iris)
    if(input$type=="LDA"){
      library(MASS)
      fit=lda(Species~.,data=iris)
      iris_proj=as.matrix(iris[,-5])%*%(fit$scaling)
    }else{
      pr.out = prcomp(iris[,-5], scale = TRUE)
      iris_proj=pr.out$x[,1:2]
    }
    cols<- c("steelblue1", "hotpink", "mediumpurple")
    pchs<-c(1,2,3)
    ########## projection plot
    plot(iris_proj[1:50,1],iris_proj[1:50,2],pch=1,
         xlim=c(-12,10),xlab=" ",ylab=" ",
         cex.lab=1.5, cex=1.5,
         cex.axis=1.5,  font.lab=2,col="steelblue1")
    if(input$type=="LDA"){
         title(xlab="LD1",ylab="LD2")
    }else{
      title(xlab="PC1",ylab="PC2")
    }
    points(iris_proj[51:100,1],iris_proj[51:100,2],pch=2,col="hotpink")
    points(iris_proj[101:150,1],iris_proj[101:150,2],pch=3,col="mediumpurple")
    legend("bottomright",legend=c("setosa","versicolor","virginica"),
           col=cols,pch=pchs,cex=1,text.font=2)
  })
}
shinyApp(ui = ui, server = server)
```

4 LDA as a dimension reduction/distance metric learning method

Load the German Credit data.

```r
library(caret)
#load german credit data from caret package
data(GermanCredit)
# classify two status: good or bad
## Show the first 10 columns
str(GermanCredit[, 1:10])
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

```r
## Delete two variables where all values are the same for both classes
GermanCredit[,c("Purpose.Vacation","Personal.Female.Single")] <- list(NULL)
```

Now we divide the dataset to training and test sets.

```r
# create training and test sets
set.seed(12)
trainIndex = createDataPartition(GermanCredit$Class, p = 0.7, list = FALSE, times = 1)
train.feature=GermanCredit[trainIndex,-10] # training features
train.label=GermanCredit$Class[trainIndex] # training labels
test.feature=GermanCredit[-trainIndex,-10] # test features
test.label=GermanCredit$Class[-trainIndex] # test labels
```

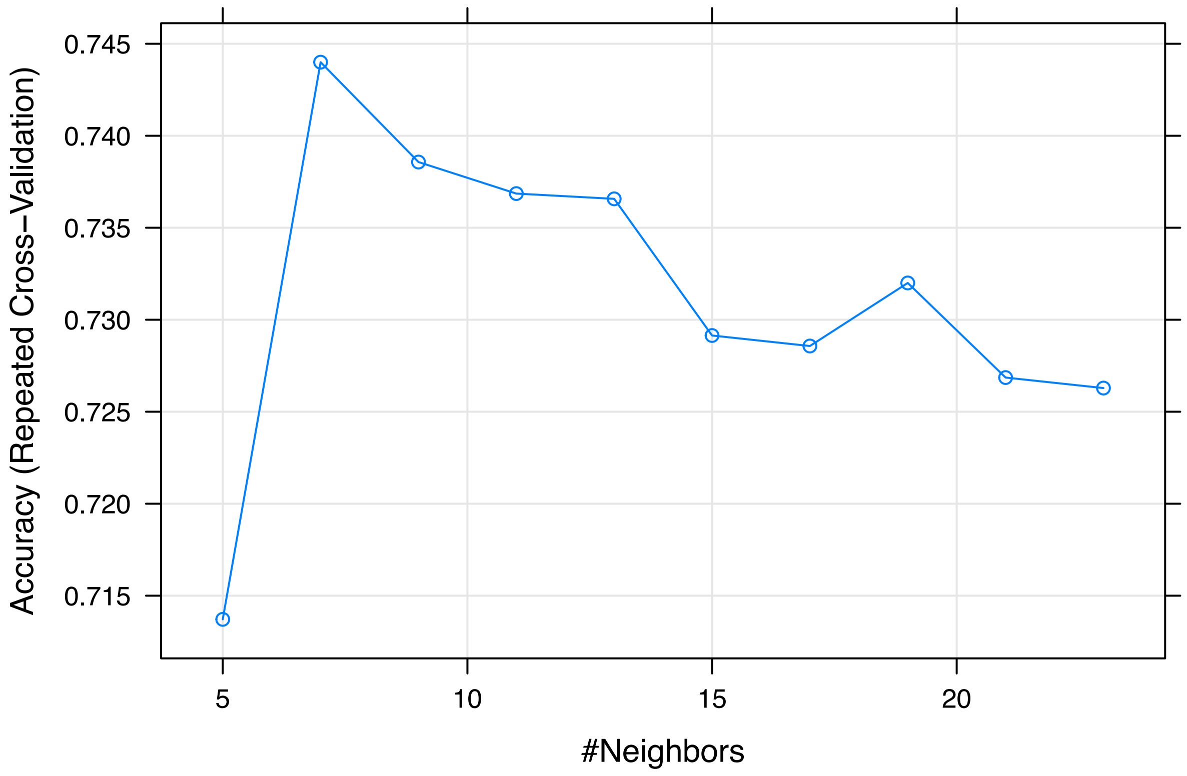
Train the kNN classifier and tune the value of `k` by 10-fold CV.

```r
#### set up train control
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated five times
  repeats = 5)
#### training process
```

```r
set.seed(5)
knnFit=train(train.feature,train.label, method = "knn",
              trControl = fitControl,
              metric = "Accuracy",
              preProcess = c("center","scale"),
              tuneLength=10)
knnFit
```

```
## k-Nearest Neighbors
##
## 700 samples
##  59 predictor
##   2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 630, 630, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7137143  0.2393261
##    7  0.7440000  0.3036671
##    9  0.7385714  0.2722017
##   11  0.7368571  0.2558584
##   13  0.7365714  0.2392079
##   15  0.7291429  0.2097804
##   17  0.7285714  0.1969141
##   19  0.7320000  0.2027894
##   21  0.7268571  0.1811377
##   23  0.7262857  0.1696587
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```r
plot(knnFit)
```



The test set is then classified as follows.

```
#### test process
pred=predict(knnFit,test.feature)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7066667
```

What if we use LDA to reduce the dimension first?

```
fit=lda(train.feature,train.label)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
train.feature.proj=as.matrix(train.feature)%*%(fit$scaling)
test.feature.proj=as.matrix(test.feature)%*%(fit$scaling)
```

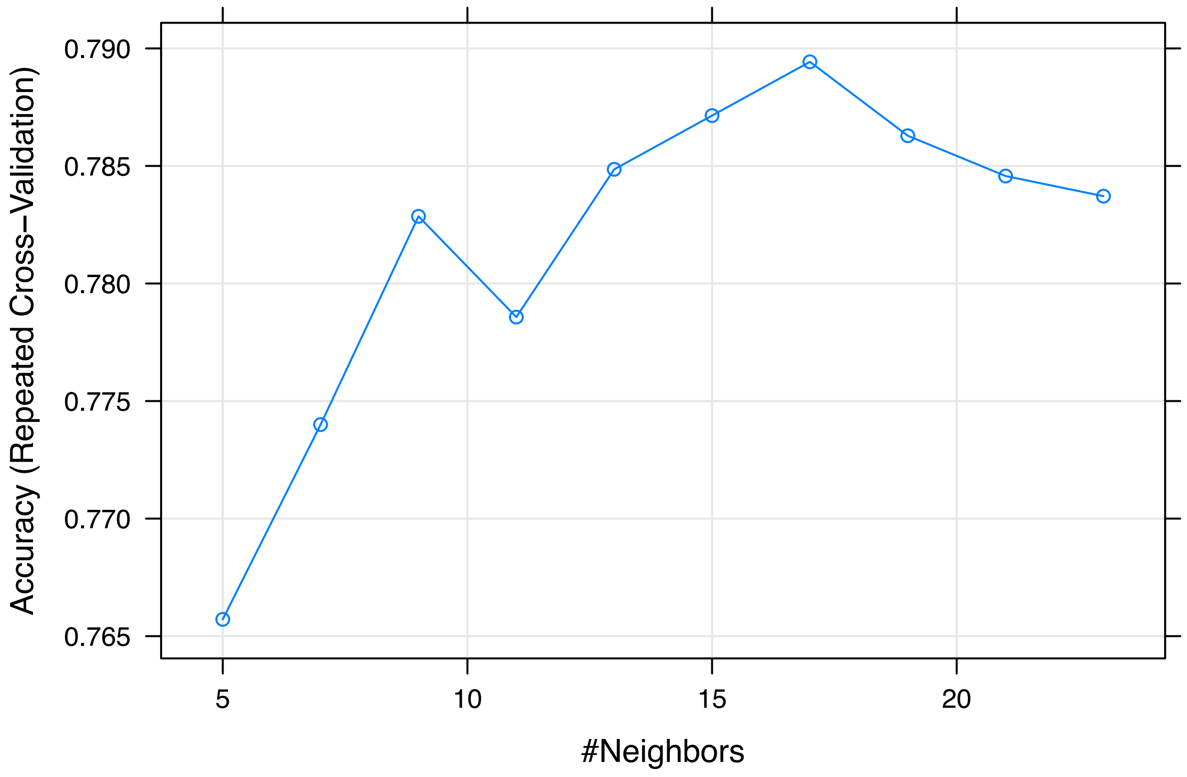
Then we apply kNN on the projected features

```
#### set up train control
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated five times
  repeats = 5)
#### training process
set.seed(5)
knnFit=train(train.feature.proj,train.label, method = "knn",
              trControl = fitControl,
              metric = "Accuracy",
              preProcess = c("center","scale"),
              tuneLength=10)
```

```r
knnFit
```

```
## k-Nearest Neighbors
##
## 700 samples
##   1 predictor
##   2 classes: 'Bad', 'Good'
##
## Pre-processing: centered (1), scaled (1)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 630, 630, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7657143  0.4145258
##    7  0.7740000  0.4306082
##    9  0.7828571  0.4507894
##   11  0.7785714  0.4467159
##   13  0.7848571  0.4656049
##   15  0.7871429  0.4672955
##   17  0.7894286  0.4732363
##   19  0.7862857  0.4635357
##   21  0.7845714  0.4610787
##   23  0.7837143  0.4610243
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 17.
```

```r
plot(knnFit)
```



The test set is then classified as follows.

```
#### test process
pred=predict(knnFit,test.feature.proj)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7533333
```

We observe a clear increase in the classification accuracy. However, this is only on this specific training/test split. We can see a more reliable result on several different training/test splits.

More real data examples

Fatty Acid Composition Data

Let's try to use LDA on the fatty acid composition data, which aims to classify seven commercial oils. First, we load the data from the package.

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
#load oil data from caret package
data(oil)
```

Then, we divide the data to training and test sets.

```
# create training and test sets
set.seed(32)
trainIndex = createDataPartition(oilType, p = 0.7, list = FALSE, times = 1)
train.feature=fattyAcids[trainIndex,] # training features
train.label=oilType[trainIndex] # training labels
test.feature=fattyAcids[-trainIndex,] # test features
test.label=oilType[-trainIndex] # test labels
```

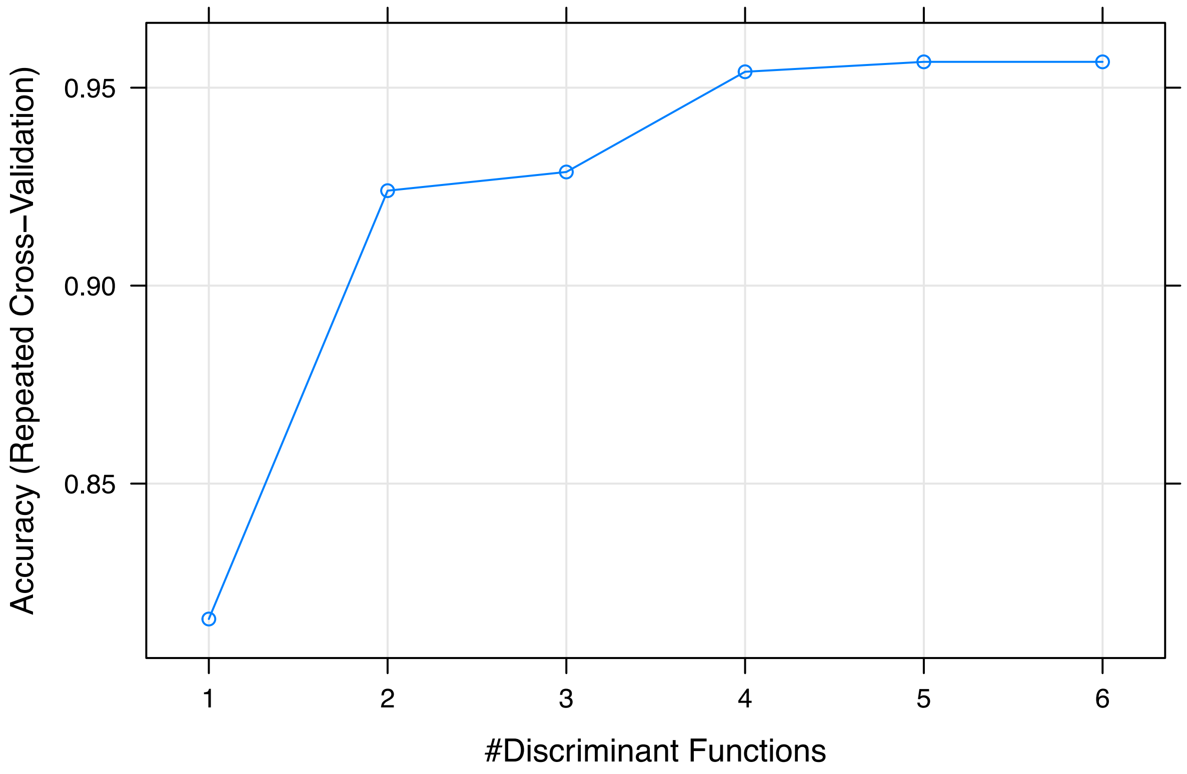
The parameter in LDA is the number of linear discriminants. For binary classification, we cannot tune this parameter, because we can only have one (2-1) linear discriminants. However, for multi-class classification, we need to tune this parameter, because the maximum number of linear discriminants is C-1 where C is the number of classes. Here we can tune it via 10-fold CV.

```
#### set up train control
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 10,
  ## repeated five times
  repeats = 5)
#### training process
set.seed(5)
ldaFit1=train(train.feature,train.label, method = "lda2",
            trControl = fitControl,
            metric = "Accuracy",
            tuneLength=10)
ldaFit1
```

```
## Linear Discriminant Analysis
##
## 70 samples
##  7 predictor
##  7 classes: 'A', 'B', 'C', 'D', 'E', 'F', 'G'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 62, 63, 62, 63, 62, 65, ...
## Resampling results across tuning parameters:
##
```

```
##   dimen  Accuracy   Kappa
##   1      0.8157937  0.7490790
##   2      0.9239841  0.9019950
##   3      0.9287063  0.9076874
##   4      0.9540238  0.9398278
##   5      0.9565238  0.9430278
##   6      0.9565238  0.9430278
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was dimen = 5.
```

```
plot(ldaFit1)
```



Based on the trained model, we can predict the labels of the test instances.

```
#### test process
pred=predict(ldaFit1,test.feature)
acc=mean(pred==test.label)
acc
```

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

```
table(pred,test.label)
```

```
##     test.label
## pred  A  B  C  D  E  F  G
##    A 11  0  0  0  0  0  0
##    B  0  7  0  0  0  0  0
##    C  0  0  0  0  0  0  0
##    D  0  0  0  2  0  0  0
##    E  0  0  0  0  3  0  0
##    F  0  0  0  0  0  3  0
##    G  0  0  0  0  0  0  0
```

German Credit Data

Now we try to classify the German credit data by LDA. Since we just need to distinguish between two classes, good or bad, there is no parameter to be tuned for this task.

```
library(caret)
#load german credit data from caret package
data(GermanCredit)
# classify two classes: good or bad
## Show the first 10 columns
str(GermanCredit[, 1:10])
```

```
## 'data.frame':    1000 obs. of  10 variables:
##  $ Duration                 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                   : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ InstallmentRatePercentage: int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration        : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                      : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits    : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance  : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                    : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
```

```
## Delete two variables where all values are the same for both classes
GermanCredit[,c("Purpose.Vacation","Personal.Female.Single")] <- list(NULL)
# create training and test sets
set.seed(12)
trainIndex = createDataPartition(GermanCredit$Class, p = 0.7, list = FALSE, times = 1)
train.feature=GermanCredit[trainIndex,-10] # training features
train.label=GermanCredit$Class[trainIndex] # training labels
test.feature=GermanCredit[-trainIndex,-10] # test features
test.label=GermanCredit$Class[-trainIndex] # test labels
#### training process
ldaFit=train(train.feature,train.label, method = "lda",
             trControl = trainControl(method = "none"))
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
ldaFit$finalModel
```

```
## Call:
## lda(x, y)
##
## Prior probabilities of groups:
##  Bad Good
##  0.3  0.7
##
## Group means:
##      Duration   Amount InstallmentRatePercentage ResidenceDuration
## Bad  24.72381 4109.448                  3.119048          2.819048
## Good 18.63265 2852.037                  2.975510          2.851020
##           Age NumberExistingCredits NumberPeopleMaintenance Telephone
## Bad  33.55714              1.361905                1.142857 0.6380952
## Good 36.48571              1.424490                1.171429 0.5857143
##      ForeignWorker CheckingAccountStatus.lt.0
## Bad      0.9809524                  0.4571429
## Good     0.9469388                  0.2020408
##      CheckingAccountStatus.0.to.200 CheckingAccountStatus.gt.200
## Bad                        0.352381                   0.02380952
## Good                       0.244898                   0.06938776
##      CheckingAccountStatus.none CreditHistory.NoCredit.AllPaid
## Bad                   0.1666667                     0.10000000
## Good                  0.4836735                     0.02244898
##      CreditHistory.ThisBank.AllPaid CreditHistory.PaidDuly
## Bad                      0.09523810              0.5571429
## Good                     0.03061224              0.5244898
##      CreditHistory.Delay CreditHistory.Critical Purpose.NewCar
## Bad           0.08571429              0.1619048      0.2952381
## Good          0.07755102              0.3448980      0.2122449
##      Purpose.UsedCar Purpose.Furniture.Equipment Purpose.Radio.Television
## Bad       0.04285714                   0.1904762                0.2142857
## Good      0.10816327                   0.1877551                0.3163265
##      Purpose.DomesticAppliance Purpose.Repairs Purpose.Education
## Bad                 0.01428571      0.01904762        0.07619048
## Good                0.01224490      0.01632653        0.04081633
##      Purpose.Retraining Purpose.Business Purpose.Other
## Bad         0.004761905       0.12380952   0.019047619
## Good        0.010204082       0.08979592   0.006122449
##      SavingsAccountBonds.lt.100 SavingsAccountBonds.100.to.500
## Bad                   0.7238095                      0.1285714
## Good                  0.5489796                      0.1040816
##      SavingsAccountBonds.500.to.1000 SavingsAccountBonds.gt.1000
## Bad                       0.03809524                  0.01904762
## Good                      0.07755102                  0.05918367
##      SavingsAccountBonds.Unknown EmploymentDuration.lt.1
## Bad                   0.09047619               0.2333333
## Good                  0.21020408               0.1367347
##      EmploymentDuration.1.to.4 EmploymentDuration.4.to.7
## Bad                  0.3476190                 0.1285714
## Good                 0.3204082                 0.2163265
##      EmploymentDuration.gt.7 EmploymentDuration.Unemployed
## Bad                0.2142857                    0.07619048
## Good               0.2714286                    0.05510204
##      Personal.Male.Divorced.Seperated Personal.Female.NotSingle
## Bad                        0.04761905                 0.3428571
## Good                       0.04081633                 0.2938776
##      Personal.Male.Single Personal.Male.Married.Widowed
## Bad             0.5380952                    0.07142857
## Good            0.5857143                    0.07959184
##      OtherDebtorsGuarantors.None OtherDebtorsGuarantors.CoApplicant
## Bad                    0.9095238                         0.05238095
## Good                   0.9122449                         0.02244898
##      OtherDebtorsGuarantors.Guarantor Property.RealEstate
## Bad                        0.03809524           0.2000000
## Good                       0.06530612           0.3183673
##      Property.Insurance Property.CarOther Property.Unknown
## Bad           0.2476190         0.3285714        0.2238095
## Good          0.2285714         0.3306122        0.1224490
##      OtherInstallmentPlans.Bank OtherInstallmentPlans.Stores
## Bad                   0.2095238                   0.07142857
## Good                  0.1142857                   0.04081633
##      OtherInstallmentPlans.None Housing.Rent Housing.Own Housing.ForFree
## Bad                   0.7190476    0.2333333   0.6238095      0.14285714
## Good                  0.8448980    0.1530612   0.7551020      0.09183673
##      Job.UnemployedUnskilled Job.UnskilledResident Job.SkilledEmployee
## Bad               0.02380952             0.1714286           0.6380952
## Good              0.01836735             0.2081633           0.6510204
##      Job.Management.SelfEmp.HighlyQualified
## Bad                               0.1666667
## Good                              0.1224490
##
## Coefficients of linear discriminants:
##                                          LD1
## Duration                       -0.0216856021
## Amount                         -0.0001254021
## InstallmentRatePercentage      -0.1477892602
## ResidenceDuration               0.0059335640
## Age                             0.0129255354
## NumberExistingCredits          -0.2356809987
## NumberPeopleMaintenance         0.1039756437
## Telephone                      -0.3037740810
## ForeignWorker                  -0.7555334926
## CheckingAccountStatus.lt.0     -0.6802651221
## CheckingAccountStatus.0.to.200 -0.1566746314
## CheckingAccountStatus.gt.200    0.6204429507
## CheckingAccountStatus.none      0.5822805598
## CreditHistory.NoCredit.AllPaid -0.8002929257
## CreditHistory.ThisBank.AllPaid -0.8999539610
## CreditHistory.PaidDuly         -0.0910125573
## CreditHistory.Delay             0.2346992535
## CreditHistory.Critical          0.4083883730
## Purpose.NewCar                 -0.5229639366
## Purpose.UsedCar                 0.7796256642
## Purpose.Furniture.Equipment     0.2419758273
## Purpose.Radio.Television        0.1226363293
## Purpose.DomesticAppliance      -0.5208304623
## Purpose.Repairs                -0.1380512748
## Purpose.Education              -0.6722419773
## Purpose.Retraining              0.4130997040
## Purpose.Business                0.0190668746
## Purpose.Other                   0.6246648314
## SavingsAccountBonds.lt.100     -0.2836749290
## SavingsAccountBonds.100.to.500 -0.0742162810
## SavingsAccountBonds.500.to.1000 0.0751536157
## SavingsAccountBonds.gt.1000     0.4764658400
## SavingsAccountBonds.Unknown     0.3384464680
## EmploymentDuration.lt.1        -0.2307035092
## EmploymentDuration.1.to.4      -0.0926359542
## EmploymentDuration.4.to.7       0.4548014521
## EmploymentDuration.gt.7        -0.0577882211
## EmploymentDuration.Unemployed  -0.1124774496
## Personal.Male.Divorced.Seperated       -0.1276359750
## Personal.Female.NotSingle              -0.0990109104
## Personal.Male.Single                    0.0886405260
## Personal.Male.Married.Widowed           0.0652361362
## OtherDebtorsGuarantors.None            -0.1492637088
## OtherDebtorsGuarantors.CoApplicant     -0.6843004264
## OtherDebtorsGuarantors.Guarantor        0.6096420316
## Property.RealEstate                     0.0609739066
## Property.Insurance                      0.0119492109
## Property.CarOther                       0.1110520535
## Property.Unknown                       -0.3053798731
## OtherInstallmentPlans.Bank             -0.2787415310
## OtherInstallmentPlans.Stores           -0.2033191150
## OtherInstallmentPlans.None              0.2837016837
## Housing.Rent                           -0.2718830803
## Housing.Own                             0.0062887175
## Housing.ForFree                         0.3995714048
## Job.UnemployedUnskilled                -0.1791354529
## Job.UnskilledResident                   0.0737468347
## Job.SkilledEmployee                     0.0089103399
## Job.Management.SelfEmp.HighlyQualified -0.0870595478
```

Now we can predict the labels of test instances:

```
#### test process
pred=predict(ldaFit,test.feature)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7433333
```

```
table(pred,test.label)
```

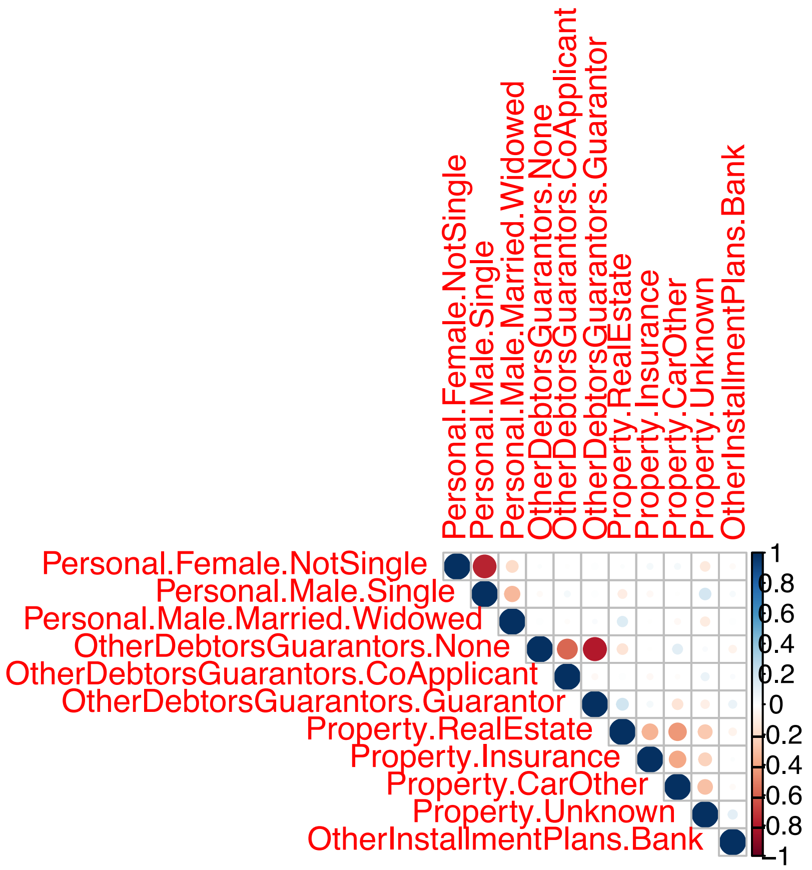
```
##       test.label
## pred   Bad Good
##   Bad   40   27
##   Good  50  183
```

We can see that there is a warning for variable colinearity given by the `lda` function. This means that there are variables with correlations of 1 or -1. Now let's have a look at variable colinearity. We can do this by plot the pairwise correlations. Given that there are 59 variables, a plot of all variables will be too crowded. We can creat a plot that contain a subset of the variables.

```
#### check variable colinearity
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
corrplot(round(cor(train.feature[,40:50]),2), type = "upper")
```



Pima Indians Diabetes Data

We now try to classify the diabetes data by LDA. Similarly to the German credit data, we only have two classes here, so there is no parameter to be tuned in LDA.

```
library(mlbench)
library(caret)
#load Pima Indians Diabetes data from mlbench package
data(PimaIndiansDiabetes)
dim(PimaIndiansDiabetes)
```

```
## [1] 768   9
```

```
levels(PimaIndiansDiabetes$diabetes)
```

```
## [1] "neg" "pos"
```

```
table(PimaIndiansDiabetes$diabetes)
```

```
##
## neg pos
## 500 268
```

```
# create training and test sets
set.seed(76)
trainIndex = createDataPartition(PimaIndiansDiabetes$diabetes, p = 0.7, list = FALSE, times = 1)
train.feature=PimaIndiansDiabetes[trainIndex,-9] # training features
train.label=PimaIndiansDiabetes$diabetes[trainIndex] # training labels
```

```
test.feature=PimaIndiansDiabetes[-trainIndex,-9] # test features
test.label=PimaIndiansDiabetes$diabetes[-trainIndex] # test labels
#### training process
ldaFit=train(train.feature,train.label, method = "lda",trControl = trainControl(method = "none"))
ldaFit$finalModel
```

```
## Call:
## lda(x, y)
##
## Prior probabilities of groups:
##       neg       pos
## 0.6505576 0.3494424
##
## Group means:
##     pregnant  glucose pressure  triceps   insulin     mass  pedigree
## neg 3.262857 111.1029 68.19714 18.98857  68.71143 29.79971 0.4401314
## pos 4.787234 142.4681 71.93617 23.54787 103.90957 35.40479 0.5463564
##          age
## neg 31.36286
## pos 36.63298
##
## Coefficients of linear discriminants:
##                    LD1
## pregnant  0.1105303578
## glucose   0.0269578458
## pressure -0.0104081197
## triceps   0.0071594610
## insulin  -0.0008296858
## mass      0.0603986613
## pedigree  0.4637060543
## age       0.0070345560
```

The labels of the test instances can be predicted via the trained LDA model.

```
#### test process
pred=predict(ldaFit,test.feature)
acc=mean(pred==test.label)
acc
```

```
## [1] 0.7652174
```

```
table(pred,test.label)
```

```
##      test.label
## pred  neg pos
##   neg 131  35
##   pos  19  45
```