Exercises: LDA, QDA

2022-03-03

In this exercise, 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
- How to use LDA as a dimension reduction
- How to apply QDA

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            5.008       3.476        1.416       0.240
## versicolor        5.844       2.720        4.088       1.256
## virginica         6.604       2.968        5.604       2.052
##
## Coefficients of linear discriminants:
##                     LD1         LD2
## Sepal.Length  0.7829268  0.06131406
## Sepal.Width   1.1769097  1.92555482
## Petal.Length -1.8186495 -1.05081846
## Petal.Width  -3.3514447  3.05271428
##
## Proportion of trace:
##    LD1    LD2
## 0.9882 0.0118
```

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:

```r
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 virginica  versicolor
## [37] versicolor 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  versicolor virginica  virginica
## [67] virginica  virginica  virginica  virginica  virginica  virginica
## [73] virginica  virginica  virginica
## Levels: setosa versicolor virginica
##
## $posterior
##          setosa   versicolor    virginica
## 3   1.000000e+00 1.334897e-15 1.848684e-36
## 4   1.000000e+00 2.447817e-13 2.463790e-33
## 5   1.000000e+00 7.282389e-18 2.373931e-39
## 6   1.000000e+00 2.872670e-16 3.847809e-36
## 7   1.000000e+00 1.495127e-14 2.332214e-34
```

```
## 12  1.000000e+00 8.808361e-15 3.875245e-35
## 13  1.000000e+00 3.313284e-15 1.594439e-36
## 17  1.000000e+00 3.717092e-19 3.250212e-40
## 18  1.000000e+00 1.540017e-16 3.229367e-37
## 19  1.000000e+00 1.046696e-17 9.373525e-39
## 21  1.000000e+00 8.224737e-16 8.928598e-37
## 25  1.000000e+00 1.291091e-12 4.398206e-32
## 26  1.000000e+00 2.952986e-13 1.926718e-33
## 27  1.000000e+00 3.473358e-13 3.670272e-32
## 28  1.000000e+00 3.370448e-17 1.425486e-38
## 29  1.000000e+00 2.152739e-17 6.040701e-39
## 30  1.000000e+00 1.956634e-13 2.098162e-33
## 32  1.000000e+00 4.475757e-15 5.967768e-35
## 35  1.000000e+00 3.257482e-14 1.157932e-34
## 36  1.000000e+00 3.369227e-17 8.329455e-39
## 40  1.000000e+00 2.223196e-16 1.746035e-37
## 43  1.000000e+00 1.003101e-14 3.933538e-35
## 45  1.000000e+00 2.020915e-13 3.937631e-32
## 46  1.000000e+00 5.012384e-13 1.159545e-32
## 50  1.000000e+00 2.781295e-16 2.050301e-37
## 52  3.666686e-18 9.990147e-01 9.853369e-04
## 53  4.051409e-20 9.965129e-01 3.487126e-03
## 55  1.055784e-20 9.959227e-01 4.077281e-03
## 57  1.793805e-20 9.719607e-01 2.803931e-02
## 60  7.018607e-19 9.994061e-01 5.938836e-04
## 64  5.638167e-21 9.964475e-01 3.552469e-03
## 67  1.454267e-21 9.734554e-01 2.654463e-02
## 68  3.264620e-14 9.999999e-01 1.255260e-07
## 69  4.951128e-24 9.709066e-01 2.909338e-02
## 71  3.026108e-26 6.482626e-02 9.351737e-01
## 72  2.340958e-15 9.999943e-01 5.690073e-06
## 73  4.316734e-25 8.745249e-01 1.254751e-01
## 74  2.541572e-19 9.999027e-01 9.733414e-05
## 76  8.051390e-17 9.999381e-01 6.189220e-05
## 78  2.104860e-24 5.335338e-01 4.664662e-01
## 81  1.355599e-15 9.999993e-01 7.197238e-07
## 83  5.999121e-15 9.999985e-01 1.541761e-06
## 85  3.692484e-22 9.482475e-01 5.175253e-02
## 87  2.940740e-19 9.982137e-01 1.786273e-03
## 88  2.682559e-20 9.998304e-01 1.696462e-04
## 89  1.746546e-16 9.999630e-01 3.702808e-05
## 90  1.085386e-18 9.998968e-01 1.031572e-04
## 91  5.816860e-20 9.998264e-01 1.736265e-04
## 92  1.003304e-19 9.986290e-01 1.371039e-03
## 95  8.672224e-19 9.998344e-01 1.656367e-04
## 103 3.440914e-39 4.270775e-06 9.999957e-01
## 104 1.575334e-34 5.230154e-04 9.994770e-01
## 106 4.184064e-44 2.040551e-07 9.999998e-01
## 109 2.236824e-37 1.804057e-04 9.998196e-01
## 110 1.235962e-44 3.416162e-09 1.000000e+00
## 114 1.615178e-37 2.742304e-05 9.999726e-01
## 117 5.565138e-32 2.712325e-03 9.972877e-01
## 121 1.309984e-40 2.993133e-07 9.999997e-01
## 126 1.105728e-32 1.750689e-03 9.982493e-01
```

```
## 127 1.691626e-27 6.202568e-02 9.379743e-01
## 128 1.171439e-27 3.911328e-02 9.608867e-01
## 130 3.764761e-28 1.607386e-01 8.392614e-01
## 133 1.868232e-42 2.440635e-07 9.999998e-01
## 134 5.432351e-25 8.010741e-01 1.989259e-01
## 135 9.919516e-30 2.108461e-01 7.891539e-01
## 137 6.915290e-43 1.835217e-08 1.000000e+00
## 139 4.341998e-27 5.385664e-02 9.461434e-01
## 140 2.445882e-34 8.487354e-05 9.999151e-01
## 141 4.720866e-43 3.250943e-08 1.000000e+00
## 144 4.344853e-43 5.405573e-08 9.999999e-01
## 145 1.036946e-44 4.083273e-09 1.000000e+00
## 146 1.078232e-37 2.630566e-06 9.999974e-01
## 147 6.226611e-33 1.522161e-03 9.984778e-01
## 148 8.704867e-33 4.378766e-04 9.995621e-01
## 150 1.451661e-30 5.169425e-03 9.948306e-01
##
## $x
##            LD1           LD2
## 3    6.9581717 -2.637768e-01
## 4    6.3984581 -6.726273e-01
## 5    7.4819486  4.197575e-01
## 6    6.9323085  1.317247e+00
## 7    6.5982515  3.153924e-01
## 12   6.7262514 -1.877799e-01
## 13   6.9543620 -1.053110e+00
## 17   7.6597683  1.737574e+00
## 18   7.1074059  5.386049e-01
## 19   7.3846400  8.378142e-01
## 21   7.0141426 -2.560733e-01
## 25   6.1806566 -5.030255e-01
## 26   6.4120729 -9.457390e-01
## 27   6.2125479  4.350257e-01
## 28   7.3389781  1.343830e-01
## 29   7.4031521  4.690939e-02
## 30   6.4125768 -5.790223e-01
## 32   6.7075835  5.646332e-01
## 35   6.6333362 -6.542331e-01
## 36   7.3749147 -1.403007e-01
## 40   7.1429944 -6.430386e-02
## 43   6.7232936 -2.821710e-01
## 45   6.2160096  8.961335e-01
## 46   6.2840730 -4.425667e-01
## 50   7.1288757 -1.579089e-01
## 52  -1.8874094  4.463666e-01
## 53  -2.3410967 -1.358592e-01
## 55  -2.4617455 -4.228058e-01
## 57  -2.5468855  7.278984e-01
## 60  -1.9890422 -2.647680e-01
## 64  -2.5039457 -6.651292e-01
## 67  -2.7491327  1.220440e-02
## 68  -0.5424381 -1.659229e+00
## 69  -3.2209044 -1.491451e+00
## 71  -3.8299010  1.016278e+00
```

```
## 72  -1.0134376 -4.273832e-01
## 73  -3.5169986 -1.327981e+00
## 74  -1.9513478 -1.468228e+00
## 76  -1.4491965 -1.266711e-01
## 78  -3.4675270  1.647835e-01
## 81  -0.9199387 -1.634773e+00
## 83  -0.8489971 -8.385224e-01
## 85  -2.9057181 -5.840639e-05
## 87  -2.1339522  6.204166e-02
## 88  -2.1727668 -1.798225e+00
## 89  -1.3513840 -1.780111e-01
## 90  -1.8362665 -1.041838e+00
## 91  -2.1108909 -1.574881e+00
## 92  -2.2043898 -3.674918e-01
## 95  -1.8863218 -8.607594e-01
## 103 -6.1317187  4.646582e-01
## 104 -5.3247228 -3.775173e-01
## 106 -7.0133100 -2.402577e-01
## 109 -5.8460459 -1.333377e+00
## 110 -7.0515880  2.637045e+00
## 114 -5.8443420  5.650631e-02
## 117 -4.8685816 -6.761713e-02
## 121 -6.3594812  1.658213e+00
## 126 -4.9944756 -1.649956e-01
## 127 -4.0657869  2.644506e-01
## 128 -4.0905626  5.383483e-01
## 130 -4.1958387 -9.504857e-01
## 133 -6.7046990  6.571444e-01
## 134 -3.5276556 -9.604778e-01
## 135 -4.4938032 -2.188532e+00
## 137 -6.7471348  2.416889e+00
## 139 -3.9869903  6.372988e-01
## 140 -5.2612883  1.170360e+00
## 141 -6.7870370  1.863748e+00
## 144 -6.8015038  1.441918e+00
## 145 -7.0686645  2.449048e+00
## 146 -5.8421237  1.786248e+00
## 147 -5.0394415 -2.119767e-01
## 148 -4.9932756  8.581713e-01
## 150 -4.6108778  3.159218e-01
```

`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:

```r
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            5.008       3.476        1.416       0.240
## versicolor        5.844       2.720        4.088       1.256
## virginica         6.604       2.968        5.604       2.052
##
## Coefficients of linear discriminants:
##                     LD1         LD2
## Sepal.Length  0.7829268  0.06131406
## Sepal.Width   1.1769097  1.92555482
## Petal.Length -1.8186495 -1.05081846
## Petal.Width  -3.3514447  3.05271428
##
## Proportion of trace:
##    LD1    LD2
## 0.9882 0.0118
```

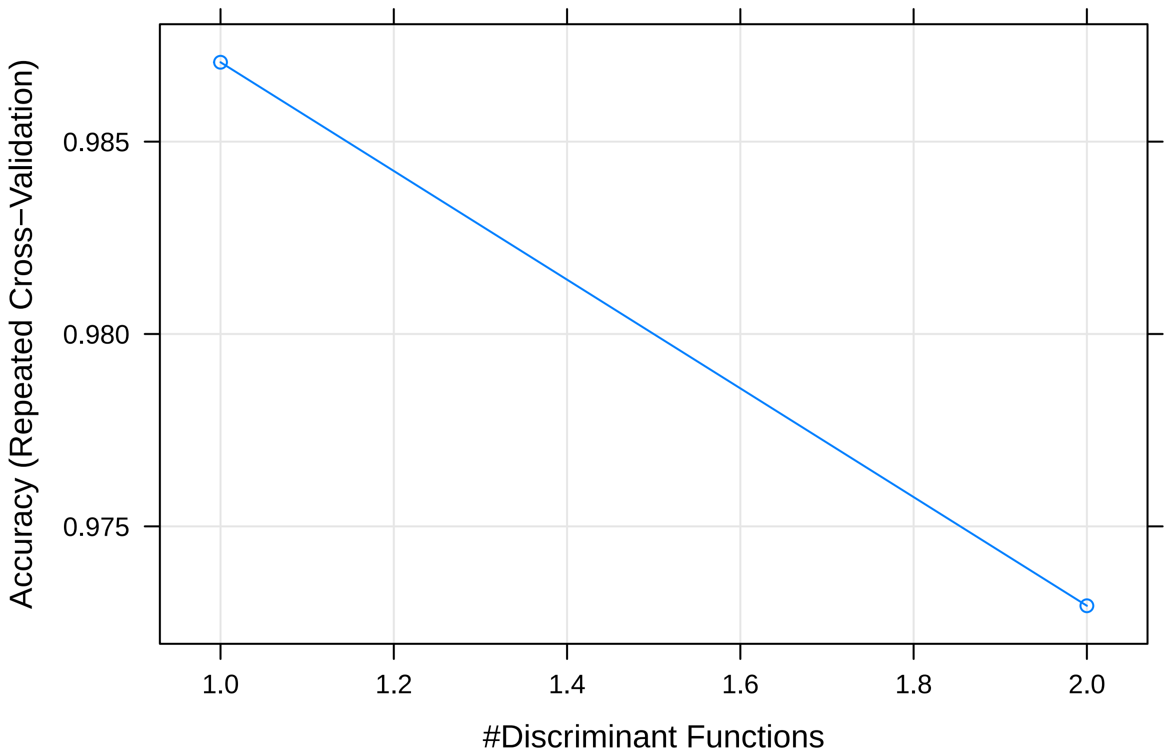
```
pred2=predict(ldaFit,test_rand[,-5])
acc2=mean(pred2==test_rand[,5])
acc2
```

```
## [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)
plot(lda2Fit)
```




```
pred3=predict(lda2Fit,test_rand[,-5])
acc3=mean(pred3==test_rand[,5])
acc3
```

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

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 LDA as a dimension reduction method

Load the German Credit data.

```
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 ...
```

```
## 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.

```
# 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
```

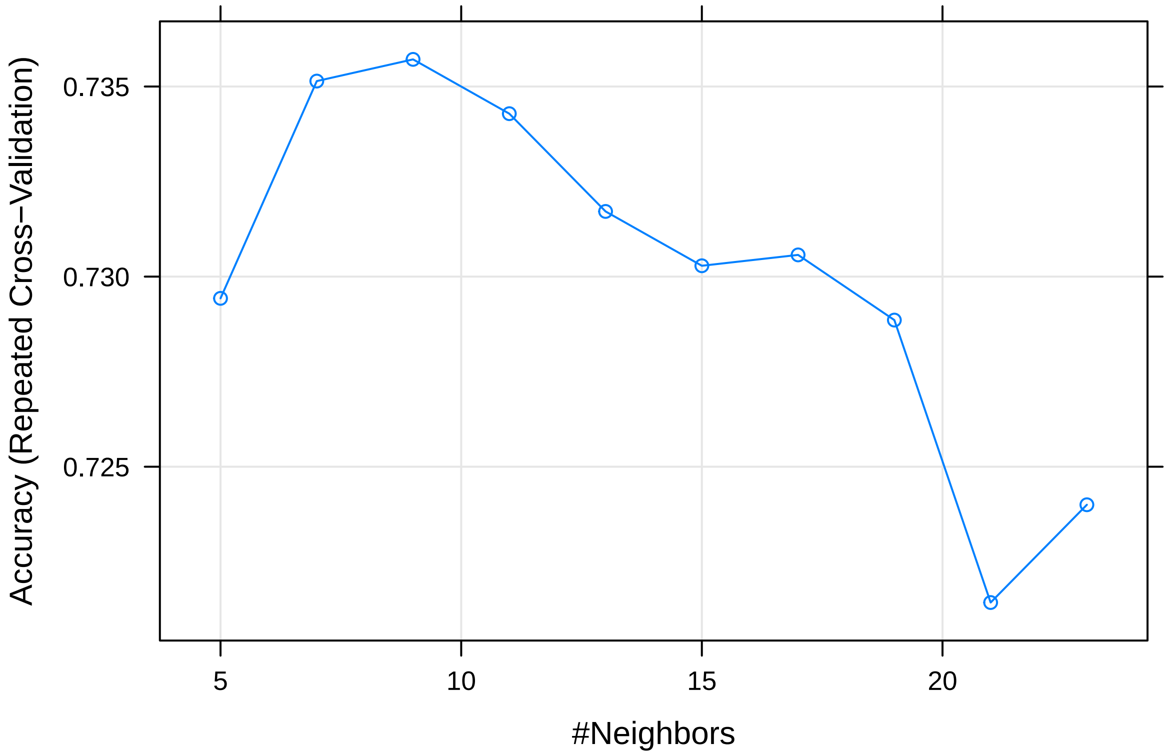
Train the kNN classifier and tune the value of `k` by 10-fold CV.

```
#### 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,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  0.7294286  0.2510201
##    7  0.7351429  0.2577826
##    9  0.7357143  0.2508425
##   11  0.7342857  0.2418045
##   13  0.7317143  0.2172966
##   15  0.7302857  0.1940371
##   17  0.7305714  0.1877088
##   19  0.7288571  0.1736438
##   21  0.7214286  0.1407893
##   23  0.7240000  0.1398291
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
plot(knnFit)
```



The test set is then classified as follows.

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

```
## [1] 0.72
```

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=predict(fit,train.feature)$x
test.feature.proj=predict(fit,test.feature)$x
```

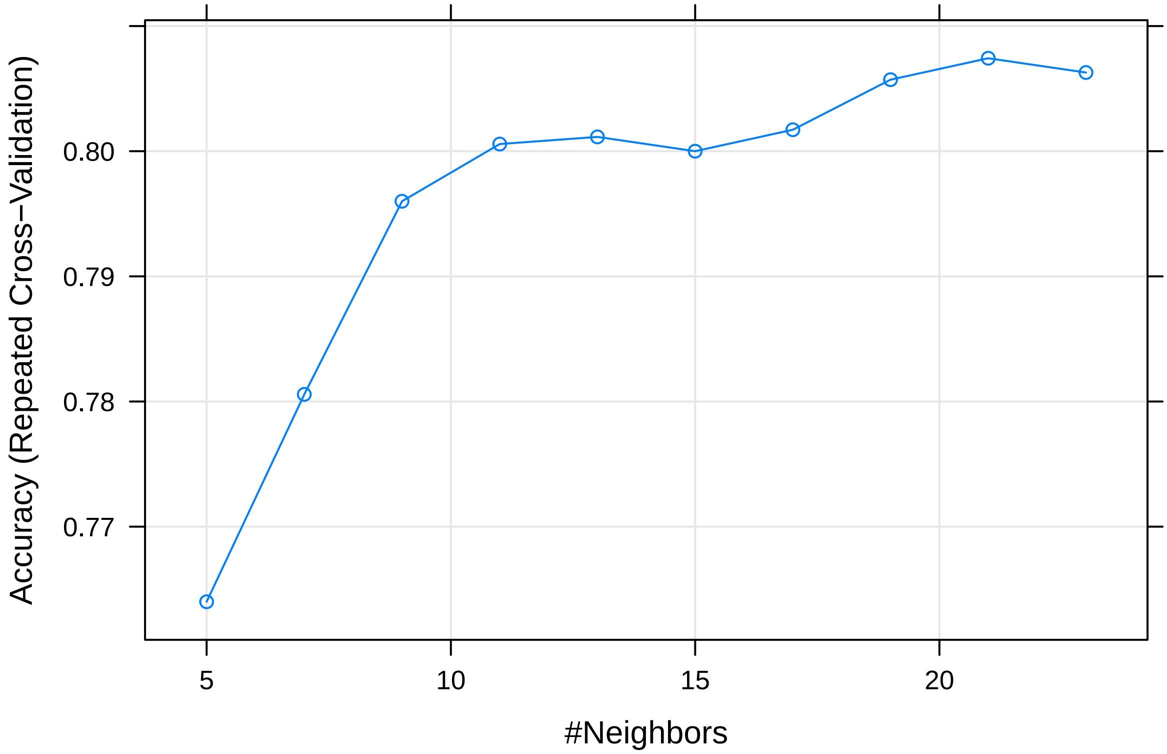
Then we apply kNN on the projected features

```
#### 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",
```

```r
                trControl = fitControl,
                metric = "Accuracy",
                preProcess = c("center","scale"),
                tuneLength=10)
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  0.7640000  0.4135291
##    7  0.7805714  0.4458202
##    9  0.7960000  0.4840862
##   11  0.8005714  0.4933397
##   13  0.8011429  0.4934285
##   15  0.8000000  0.4884783
##   17  0.8017143  0.4920930
##   19  0.8057143  0.5002162
##   21  0.8074286  0.5037261
##   23  0.8062857  0.5031759
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 21.
```

```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] 0.7266667
```

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.

4 Apply QDA by the qda function and caret package

Lastly, we are going to have a look at how to use QDA in R. Similarly to LDA, we can either use the `qda` function in the MASS library or use the caret pacakge directly.

4.1 Use the qda function

The usage of the `qda` function is very similar to that of the `lda` function: we just need to change the function name to `qda`:

```
# fit the QDA model
fit1q=qda(Species~.,data=train_rand)
fit1q
```

```
## Call:
## qda(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            5.008       3.476        1.416       0.240
## versicolor        5.844       2.720        4.088       1.256
## virginica         6.604       2.968        5.604       2.052
```

```
# predict the labels for the test set
predq=predict(fit1q,test_rand[,-5])
# calculate the classification accuracy
accq = mean(test_rand[,5]==predq$class)
accq
```

```
## [1] 0.96
```

4.2 Use the caret package

```
# fit the QDA model
qdaFit=train(train_rand[,-5],train_rand[,5],method="qda",
                    trControl=trainControl(method = "none"))
qdaFit$finalModel
```

```
## Call:
## qda(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            5.008       3.476        1.416       0.240
## versicolor        5.844       2.720        4.088       1.256
## virginica         6.604       2.968        5.604       2.052
```

```
# predict the labels for the test set
predq2=predict(qdaFit,test_rand[,-5])
# calculate the classification accuracy
accq2=mean(predq2==test_rand[,5])
accq2
```

```
## [1] 0.96
```

It is clear that the classification accuracy of QDA is less than that of LDA for this specific training/test split, which suggests that the classes can be well separated by a linear classification boundary rather than a quadratic one.