Classification

Author: Georgios Asimomitis

Introduction

The purpose of this exercise is to give a deeper insight on classifying samples using svm and knn. Initially, we load the data, we prefilter the genes reducing their dimensionality and then we apply svm and knn as well as compute their accuracy of prediction. Then we examine classification through top scoring pairs and in the last part of the exercise we tune the classification parameters to optimize classification.

1 Loading an example data set

In Bioconductor there is an example data set from the publication:

Sabina Chiaretti, Xiaochun Li, Robert Gentleman, Antonella Vitale, Marco Vignetti, Franco Mandelli, Jerome Ritz, and Robin Foa, 'Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival', Blood, 1 April 2004, Vol. 103, No. 7.

The data set contains 128 samples that were used to characterize subtypes of acute lymphoblastic leukemia and the number of features is 12625.

```
library(ALL)
data(ALL)
show(ALL)
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 12625 features, 128 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 01005 01010 ... LAL4 (128 total)
##   varLabels: cod diagnosis ... date last seen (21 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
##   pubMedIds: 14684422 16243790
## Annotation: hgu95av2
```

In this exercise we will only use samples from B-cells and the molecular biological characterizations "BCR/ABL" and "NEG". The matrix 'dataMatrix contains the expression data of 12625 genes (features) over 79 samples. Each of the samples belongs either to the class "BCR/ABL" or "NEG". The factor 'classLabels' presents the class in which each individual sample belongs. The genes of 'dataMatrix' will be used as features (variables) to classify the samples.

```
bCellSamples = grep("^B", ALL$BT)
BcrAndNegSamples = which(ALL$mol.biol %in% c("BCR/ABL", "NEG"))
samplesToUse = intersect(bCellSamples, BcrAndNegSamples)
dataMatrix = exprs(ALL[ ,samplesToUse])

head(dataMatrix)
```

```
##              01005     01010    03002    04007     04008     04010
## 1000_at   7.597323  7.479445 7.567593 7.905312  7.065914  7.474537
## 1001_at   5.046194  4.932537 4.799294 4.844565  5.147762  5.122518
## 1002_f_at 3.900466  4.208155 3.886169 3.416923  3.945869  4.150506
## 1003_s_at 5.903856  6.169024 5.860459 5.687997  6.208061  6.292713
## 1004_at   5.925260  5.912780 5.893209 5.615210  5.923487  6.046607
## 1005_at   8.570990 10.428299 9.616713 9.983809 10.063484 10.662059
##               04016    06002     08001    08011    08012    08024
## 1000_at    7.536119 7.183331  7.735545 7.591498 7.824284 7.879988
## 1001_at    5.016132 5.288943  4.633217 4.583148 4.685951 4.830464
## 1002_f_at  3.576360 3.900935  3.630190 3.609112 3.902139 3.862914
## 1003_s_at  5.665991 5.842326  5.875375 5.733157 5.762857 6.079410
## 1004_at    5.738218 5.994515  5.748350 5.922568 5.679899 6.057632
## 1005_at   11.269115 8.812869 10.165159 9.381072 8.227970 7.667445
##               09008     09017    11005    12006    12007    12012
## 1000_at    7.891793  7.756734 7.640012 7.759599 7.678636 7.464285
## 1001_at    5.999496  4.987595 4.967288 4.770481 5.456332 4.785863
## 1002_f_at  4.001606  4.048901 3.796550 3.912707 3.870893 3.930832
## 1003_s_at  5.832952  6.097900 6.094379 6.235795 5.971466 6.037364
## 1004_at    5.717497  6.210092 5.751805 5.883340 5.918456 5.725421
## 1005_at   10.206353 10.015466 9.358516 8.824348 9.262478 7.232927
##               12019    12026    14016     15001    15005    16009    20002
## 1000_at    7.652719 7.501591 7.570417  7.331509 7.455451 7.297313 7.541133
## 1001_at    5.175609 5.188992 5.258312  4.627955 5.125098 5.215707 4.964424
## 1002_f_at  3.932360 4.188444 4.028859  4.099497 3.786741 4.176687 3.789888
## 1003_s_at  6.194623 6.231228 6.348593  6.057790 6.303592 6.274415 6.160238
## 1004_at    5.969027 6.357476 6.173530  5.729543 5.977084 6.000902 6.054232
## 1005_at   10.243610 7.808452 7.557919 10.233185 8.477219 9.599149 9.221405
##              22009    22010    22011    22013    24001     24008    24010
## 1000_at   8.016818 7.862181 7.702580 7.412003 7.916169  7.296349 7.506236
## 1001_at   5.216252 5.135825 4.802946 5.222676 4.790170  5.002518 4.218220
## 1002_f_at 3.980839 3.954917 3.971934 4.109899 3.899038  3.906343 3.579385
## 1003_s_at 6.343042 6.195307 5.865581 6.243157 6.022905  5.668509 5.273965
## 1004_at   6.024327 6.114502 6.035582 5.896131 5.800271  5.437155 4.634124
## 1005_at   8.571812 8.841628 8.489550 8.998592 8.933302 10.826157 8.782330
##              24011     24017    24018    24022    25003     25006    26001
## 1000_at   7.144425  7.513972 7.815971 7.300980 7.845054  7.651229 7.376930
## 1001_at   5.228892  5.264158 4.899316 5.177703 5.250315  4.896195 5.123546
## 1002_f_at 3.829513  3.965467 4.058576 3.838198 4.046442  4.120495 4.131492
## 1003_s_at 5.817272  6.088179 6.387995 5.863318 6.205917  6.298788 6.118064
## 1004_at   5.552223  5.982065 5.874817 5.669051 5.931859  5.915944 6.002812
## 1005_at   7.881855 11.069535 9.102971 9.095296 8.670866 10.496309 9.046483
##              26003    27003    27004    28001    28005    28006     28007
## 1000_at   7.663977 7.329996 7.360754 7.035203 7.551734 7.538601  7.501531
## 1001_at   5.078104 5.438098 4.757900 5.005279 4.944978 4.511194  4.888814
## 1002_f_at 3.803233 3.677207 3.638739 3.800893 3.719482 3.788262  3.626149
## 1003_s_at 6.199316 5.899308 5.664813 5.732956 5.833428 5.362676  5.702537
## 1004_at   5.822230 5.718582 5.595820 5.485361 5.554058 4.986320  5.743743
## 1005_at   9.104846 6.969776 8.867644 7.067019 8.245585 7.807180 10.085771
##              28019    28021    28023    28024     28031    28035    28036
## 1000_at   7.116676 7.107979 7.427808 6.549926  7.377215 7.227516 7.407561
## 1001_at   5.275964 4.865566 5.057619 5.185277  4.778381 6.408157 5.042222
## 1002_f_at 4.192648 3.979372 3.791415 3.943834  3.657005 3.995074 3.714084
## 1003_s_at 6.196541 5.804445 5.719376 5.943116  5.939648 6.272305 5.733332
```

```
## 1004_at   5.926093 5.768851 5.478333 5.756534  5.770578 6.050495 5.651345
## 1005_at   8.097072 8.661098 9.106441 8.804075 10.607653 8.957027 8.764321
##              28037    28042    28043    28044    28047    30001    31011
## 1000_at   7.158049 7.235291 7.589310 7.988476 7.362458 7.508667 7.651676
## 1001_at   5.431469 4.686293 4.851805 4.894379 4.843868 5.587029 4.741654
## 1002_f_at 4.302001 3.677909 3.831514 3.690856 3.646990 3.765444 3.790688
## 1003_s_at 6.253362 6.098969 6.132159 6.130691 5.628370 6.078532 6.251328
## 1004_at   6.494545 6.109255 5.867806 5.592139 5.644372 5.935291 5.820374
## 1005_at   9.102879 7.008132 7.720932 7.861043 6.642728 9.075910 9.585180
##               33005    36002    37013    43001     43004    43007    43012
## 1000_at    7.486432 7.473427 7.627685 7.577529  7.600206 7.776844 7.585928
## 1001_at    4.642628 4.953122 5.358236 5.054157  4.879037 4.949908 5.057530
## 1002_f_at  3.682768 3.688162 4.008891 3.932435  4.028704 3.689141 3.891536
## 1003_s_at  5.961910 5.642185 6.314849 6.310934  6.086349 5.658127 6.363734
## 1004_at    5.810047 5.678327 6.044299 5.782177  5.817414 5.621938 5.975024
## 1005_at   11.609025 8.893931 7.950866 8.569400 10.693175 7.601647 9.377819
##               48001    49006     57001    62001    62002    62003    64001
## 1000_at    7.450666 7.004613  7.195206 7.407351 7.756195 7.913324 7.694588
## 1001_at    4.960382 4.836905  4.744006 4.930312 5.238937 5.074681 4.928159
## 1002_f_at  4.061201 3.699625  3.973128 3.734818 3.945514 3.926906 3.806746
## 1003_s_at  6.099140 5.616555  5.962672 5.730142 6.061704 6.208286 5.834263
## 1004_at    5.853644 5.704549  5.765632 5.512776 5.956554 6.028228 5.912557
## 1005_at   10.929231 9.193089 10.691061 9.108345 9.507559 9.693530 8.883131
##               64002    65005     68001    68003    84004
## 1000_at    7.583071 7.609538  7.324502 7.545120 7.679603
## 1001_at    4.804083 4.715693  5.379102 4.650231 4.795495
## 1002_f_at  4.104208 3.453649  4.066075 3.626514 3.554142
## 1003_s_at  6.340025 5.584102  6.121059 6.347044 5.471594
## 1004_at    6.056120 5.611407  6.224473 5.884682 5.505538
## 1005_at   10.320243 7.757368 11.165801 8.986872 9.984865
```

```
dim(dataMatrix)
```

```
## [1] 12625    79
```

```
classLabels = factor(ALL$mol.biol[samplesToUse])
classLabels
```

```
##  [1] BCR/ABL NEG     BCR/ABL NEG     NEG     NEG     NEG     NEG
##  [9] BCR/ABL BCR/ABL NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL
## [17] BCR/ABL BCR/ABL NEG     BCR/ABL BCR/ABL NEG     BCR/ABL NEG
## [25] BCR/ABL NEG     BCR/ABL NEG     BCR/ABL BCR/ABL NEG     BCR/ABL
## [33] BCR/ABL BCR/ABL NEG     BCR/ABL NEG     NEG     NEG     BCR/ABL
## [41] BCR/ABL BCR/ABL NEG     NEG     NEG     NEG     BCR/ABL BCR/ABL
## [49] NEG     NEG     NEG     NEG     BCR/ABL NEG     NEG     NEG
## [57] NEG     NEG     BCR/ABL BCR/ABL NEG     NEG     BCR/ABL BCR/ABL
## [65] NEG     NEG     NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL
## [73] BCR/ABL NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL
## Levels: BCR/ABL NEG
```

2 Prefiltering of Genes

In order to reduce the dimensionality, we select only the 1000 genes with the highest variance in our data set. Therefore, we remove the low variance genes from the data set. In the first place, we compute the variance

for each gene across all samples and then we sort the genes in decreasing order of variance. Then we select the first 1000 genes and reform our dataMatrix so as to contain only these 1000 high variance genes.

```
Var = apply(dataMatrix,1,var)
names(Var) = rownames(dataMatrix)
sortedVar = sort(Var,decreasing = TRUE);
highVarGenes = names(sortedVar[1:1000])
dataMatrix = dataMatrix[highVarGenes,]
dim(dataMatrix)
```

```
## [1] 1000   79
```

3 Support Vector Machine Classification

By using the dataMatrix, classLabels and a linear kernel we form our svm model by using the built in R function svm of the e1071 library. After having formed and trained the svm model we use the function predict from the same package in order to predict the class of each sample based on the model trained by svm. As expected, the error is zero, as training and prediction were performed in the same dataset.

Then, by adding the term 'cross = length(classLabels)' in the svm function, we run a cross validation to estimate the error when classifying unknown samples. Out of the summary of the output we observe that 51 support vectors were used and that the total Accuracy of classification is 81.01266 %.

```
library(e1071)
model = svm(t(dataMatrix), classLabels, kernel = "linear")

predicted = predict(model, t(dataMatrix))
table(true = classLabels, pred = predicted)
```

```
##          pred
## true      BCR/ABL NEG
##   BCR/ABL      37   0
##   NEG           0  42
```

```
model.cv = svm(t(dataMatrix), classLabels, kernel = "linear", cross = length(classLabels))
summary(model.cv)
```

```
##
## Call:
## svm.default(x = t(dataMatrix), y = classLabels, kernel = "linear",
##     cross = length(classLabels))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.001
##
## Number of Support Vectors:  51
##
##  ( 26 25 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  BCR/ABL NEG
##
## 79-fold cross-validation on training data:
##
## Total Accuracy: 81.01266
## Single Accuracies:
##  100 100 100 100 100 100 0 100 0 0 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 1
```

```
model.cv$tot.accuracy
```

```
## [1] 81.01266
```

After having used svm, we now perform the classification of our samples using the knn classifier. For that purpose, we make use of the function 'knn.cv', of the package 'class', which executes k-nearest neighbour cross-validatory classification using the method leave-one-out. As earlier, we transpose our dataMatrix so as the samples to be in the rows and the features (genes) in the columns.

After the classification we examine the Accuracy of the knn model by observing the correct classified samples. The Accuracy is 75.94937 %.

Additionally, we make use of the library 'gmodels' in order to observe analytically how the classLabels relate to the output of the model. We see that 32 out of the 37 'BCR/ABL' were correctly classified as 'BCR/ABL', whereas only 28 out of 48 NEG samples were correctly classified as 'NEG'. Thus the Accuracy equals 100*(28+32) / 79 which results in the same percentage that was computed before, as expected.

```
library(class)
model.knn = knn.cv(t(dataMatrix), classLabels, k = 5)
Accuracy = 100 * sum(classLabels == model.knn)/length(classLabels)
Accuracy
```

```
## [1] 75.94937
```

```
library(gmodels)
CrossTable(x = classLabels, y = model.knn, prop.chisq=FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  79
```

```
##
##
##               | model.knn
##  classLabels  |   BCR/ABL |       NEG | Row Total |
## --------------|-----------|-----------|-----------|
##      BCR/ABL  |        32 |         5 |        37 |
##               |     0.865 |     0.135 |     0.468 |
##               |     0.696 |     0.152 |           |
##               |     0.405 |     0.063 |           |
## --------------|-----------|-----------|-----------|
##           NEG |        14 |        28 |        42 |
##               |     0.333 |     0.667 |     0.532 |
##               |     0.304 |     0.848 |           |
##               |     0.177 |     0.354 |           |
## --------------|-----------|-----------|-----------|
## Column Total  |        46 |        33 |        79 |
##               |     0.582 |     0.418 |           |
## --------------|-----------|-----------|-----------|
##
##
```

4 Top Scoring Pairs Classification

Here we use the function tspcalc of the library 'tspair' in order to calculate the pair of genes that show the maximum difference in ranking between the two groups. Given as input the dataMatrix and the classLabels we compute tspResult and find which two genes form the pair returned by tspcalc. Then, according to tspResult, we predict the class of each of the samples and then by using 'CrossTable', of the library gmodels, we observe the relation between the number of TP, FP, TN, FN. '

```
library(tspair)

tspResult = tspcalc(dataMatrix,classLabels)
rownames(dataMatrix)[tspResult$index[1]]
```

```
## [1] "1635_at"
```

```
rownames(dataMatrix)[tspResult$index[2]]
```

```
## [1] "38661_at"
```

```
predictedLabels = predict(tspResult, dataMatrix)
table(predictedLabels, classLabels)
```

```
##                classLabels
## predictedLabels BCR/ABL NEG
##         BCR/ABL      30   4
##         NEG           7  38
```

```
CrossTable(x = classLabels, y = predictedLabels, prop.chisq=FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  79
##
##
##                | predictedLabels
##  classLabels |   BCR/ABL |       NEG | Row Total |
## -------------|-----------|-----------|-----------|
##      BCR/ABL |        30 |         7 |        37 |
##              |     0.811 |     0.189 |     0.468 |
##              |     0.882 |     0.156 |           |
##              |     0.380 |     0.089 |           |
## -------------|-----------|-----------|-----------|
##          NEG |         4 |        38 |        42 |
##              |     0.095 |     0.905 |     0.532 |
##              |     0.118 |     0.844 |           |
##              |     0.051 |     0.481 |           |
## -------------|-----------|-----------|-----------|
## Column Total |        34 |        45 |        79 |
##              |     0.430 |     0.570 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Within this frame of reference, we implement manually a leave-one-out cross validation. In particular, in each iteration we compute the tspResult of the tspcalc function for all samples except of one and then we evaluate the prediction by comparing the predicted class of the one sample left out with its corresponding true classLabel. We store each individual result in the vector 'correct.prediction' and then we compute the Accuracy.

```
correct.prediction <- rep(TRUE,ncol(dataMatrix))
for( i in 1:ncol(dataMatrix)){

 testdat <- dataMatrix[ , -i]
 testgrp <- classLabels[-i]
 tspResult <- tspcalc(testdat,testgrp)

 prediction <- predict(tspResult,dataMatrix)[i]
 correct.prediction[i] <- prediction == classLabels[i]

  }

Accuracy = 100*sum(correct.prediction[TRUE])/(length(correct.prediction))
Accuracy
```

```
## [1] 60.75949
```

5 Parameter Optimization

In this last part of the exercise, we examine the value of k for which the knn classifier works best. For that purpose we make use of the function tune.knn. As shown in the code as well, we can compute the k with the least error by using three different methods of sampling; 10-fold cross validation, bootstraping, fixed training/validation set. Cross validation and bootstraping create both 10 different sets for training with length 71 whereas the fixed set method uses a single fixed training set of 52 samples. Out of the plots below we can observe the relationship between the k value and the error per method and derive the optimal k value. These are shown below.

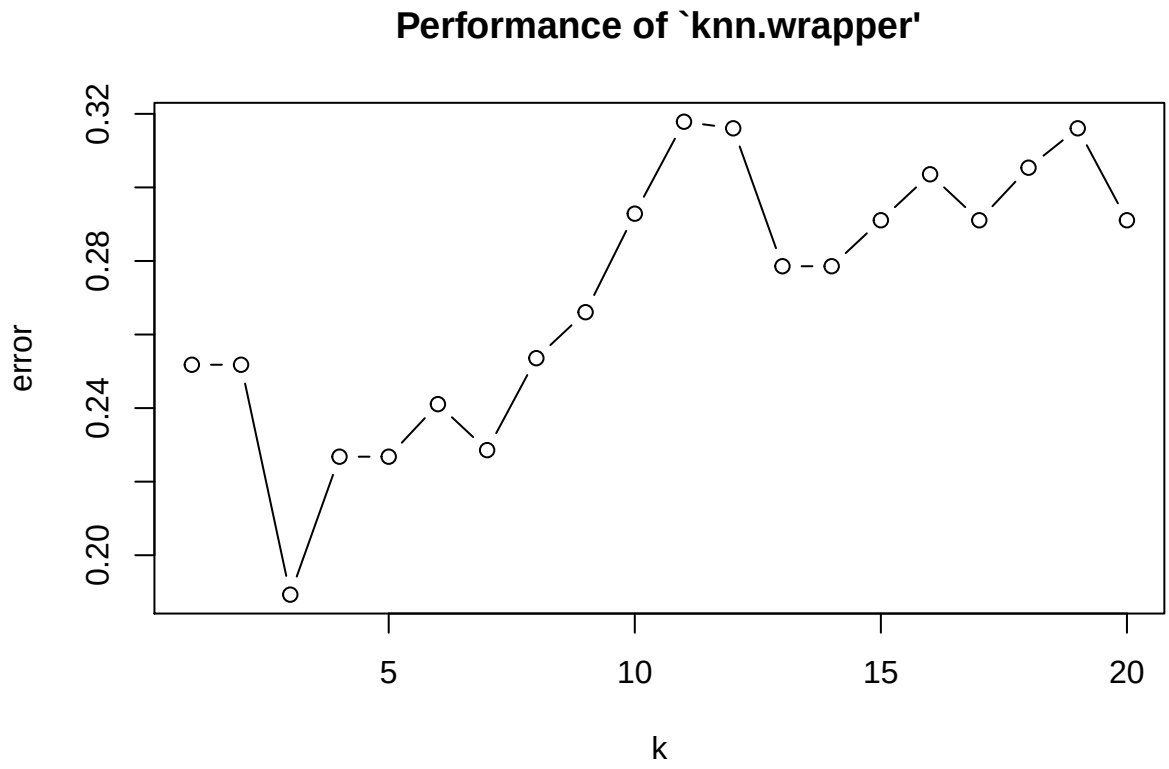
For each of the three cases we use the optimal value for k to classify the unknown samples with the method knn. In order to do that we split randomly our samples into the ones that will be used for training and the ones that will be used for testing. Using the sample function, we select 80% of our samples to be the training set and the rest 20% to be used for testing. After having separated the classLabels according to the split of the samples in the two sets, we use the knn function to perform the prediction. We also compare the prediction accuracy of the 3 sampling methods for the optimal k they tuned.

```
knn.cross <- tune.knn(x = t(dataMatrix), y = classLabels,
            k = 1:20,tunecontrol=tune.control(sampling = "cross"))
summary(knn.cross)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  k
##  3
##
## - best performance: 0.1892857
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.2517857 0.11586520
## 2   2 0.2517857 0.15440664
## 3   3 0.1892857 0.12058531
## 4   4 0.2267857 0.14061988
## 5   5 0.2267857 0.11326741
## 6   6 0.2410714 0.07253606
## 7   7 0.2285714 0.10024064
## 8   8 0.2535714 0.08409516
## 9   9 0.2660714 0.16092359
## 10 10 0.2928571 0.17111413
## 11 11 0.3178571 0.15077276
## 12 12 0.3160714 0.14605789
## 13 13 0.2785714 0.12882471
## 14 14 0.2785714 0.09824134
## 15 15 0.2910714 0.08335459
## 16 16 0.3035714 0.15726006
## 17 17 0.2910714 0.11786616
```

```
## 18 18 0.3053571 0.12555150
## 19 19 0.3160714 0.11995192
## 20 20 0.2910714 0.13177501
```

```
plot(knn.cross)
```




```
knn.cross$best.parameters[1,1]
```

```
## [1] 3
```

```
length(knn.cross$train.ind)
```

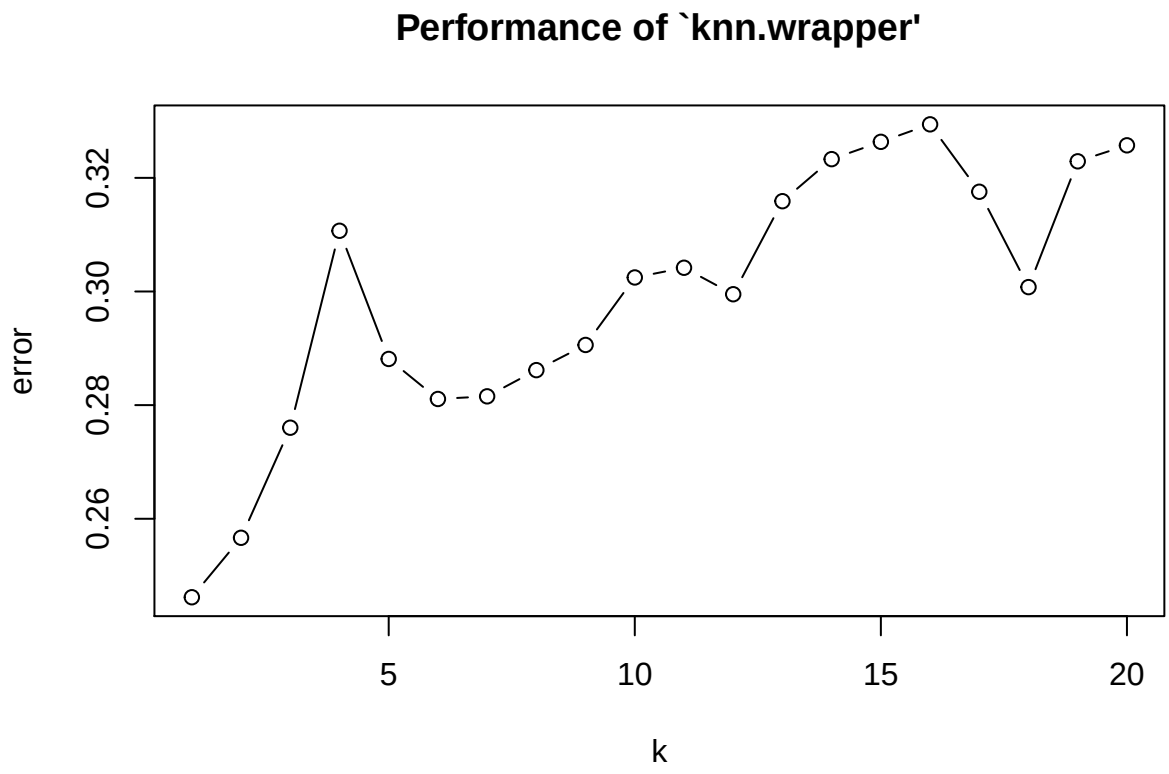
```
## [1] 10
```

```
knn.boot <- tune.knn(x = t(dataMatrix), y = classLabels,
            k = 1:20,tunecontrol = tune.control(sampling = "boot"))
summary(knn.boot)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: bootstrapping
##
```

```
## - best parameters:
##  k
##  1
##
## - best performance: 0.2461863
##
## - Detailed performance results:
##     k     error dispersion
## 1   1 0.2461863 0.07432151
## 2   2 0.2566506 0.08640316
## 3   3 0.2760257 0.09119675
## 4   4 0.3106863 0.09588073
## 5   5 0.2881283 0.08788313
## 6   6 0.2810817 0.09556158
## 7   7 0.2815445 0.11569268
## 8   8 0.2861618 0.08997249
## 9   9 0.2905881 0.10247955
## 10 10 0.3024689 0.10668916
## 11 11 0.3041765 0.11980322
## 12 12 0.2995182 0.13582121
## 13 13 0.3158909 0.12378824
## 14 14 0.3232946 0.12326072
## 15 15 0.3263421 0.13484607
## 16 16 0.3294181 0.12537432
## 17 17 0.3175526 0.10703103
## 18 18 0.3007720 0.12386417
## 19 19 0.3229106 0.12014108
## 20 20 0.3257240 0.12253265
```

```
plot(knn.boot)
```




```
knn.boot$best.parameters[1,1]
```

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

```
length(knn.boot$train.ind)
```

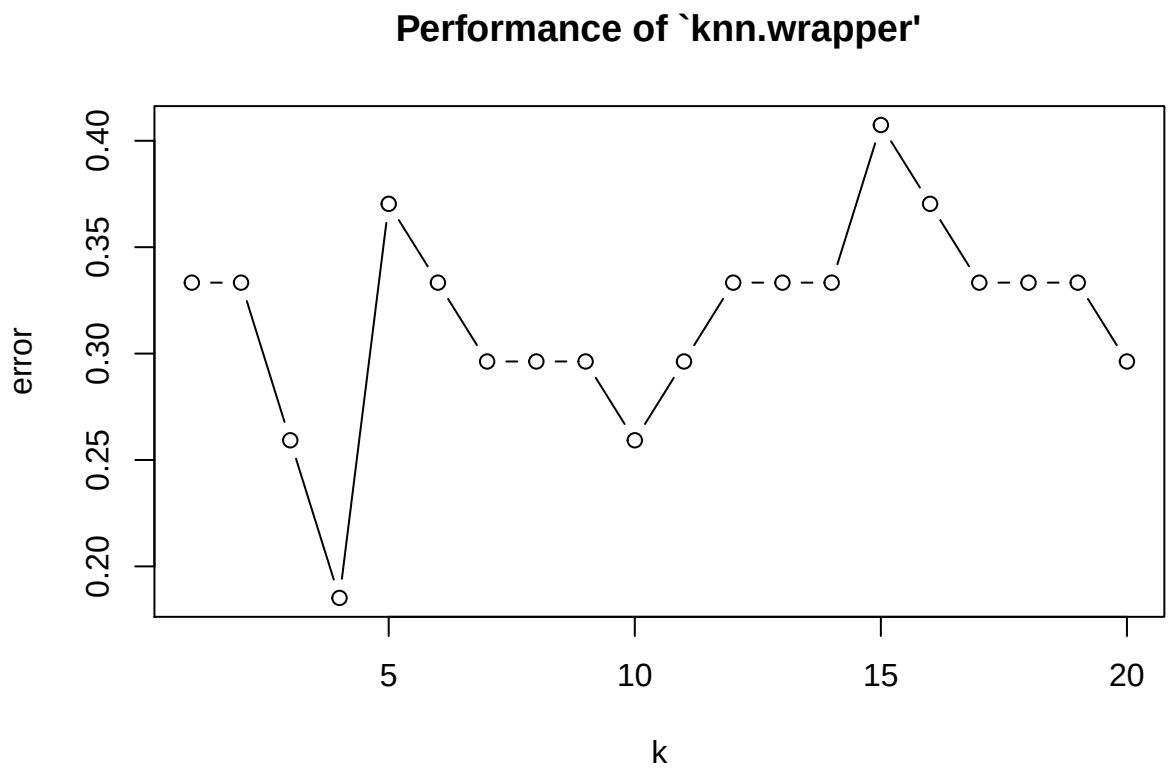
```
## [1] 10
```

```
knn.fix <- tune.knn(x = t(dataMatrix), y = classLabels,
          k = 1:20,tunecontrol=tune.control(sampling = "fix") , fix=10)
summary(knn.fix)
```

```
##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: fixed training/validation set
##
## - best parameters:
##  k
##  4
##
## - best performance: 0.1851852
##
## - Detailed performance results:
```

```
##     k     error dispersion
## 1   1 0.3333333         NA
## 2   2 0.3333333         NA
## 3   3 0.2592593         NA
## 4   4 0.1851852         NA
## 5   5 0.3703704         NA
## 6   6 0.3333333         NA
## 7   7 0.2962963         NA
## 8   8 0.2962963         NA
## 9   9 0.2962963         NA
## 10 10 0.2592593         NA
## 11 11 0.2962963         NA
## 12 12 0.3333333         NA
## 13 13 0.3333333         NA
## 14 14 0.3333333         NA
## 15 15 0.4074074         NA
## 16 16 0.3703704         NA
## 17 17 0.3333333         NA
## 18 18 0.3333333         NA
## 19 19 0.3333333         NA
## 20 20 0.2962963         NA
```

```
plot(knn.fix)
```



```
knn.fix$best.parameters[1,1]
```

```
## [1] 4
```

```
length(knn.fix$train.ind[[1]])
```

```
## [1] 52
```

```
set.seed(1234)
ind <- sample(2, ncol(dataMatrix), replace=TRUE, prob=c(0.8, 0.2))
training <- dataMatrix[,ind==1]
testing <- dataMatrix[,ind==2]
trainLabels <- classLabels[ind==1]
testLabels <- classLabels[ind==2]

pred_cross <- knn(train = t(training), test = t(testing),
              cl = trainLabels, k=knn.cross$best.parameters[1,1])
table(pred_cross, testLabels)
```

```
##           testLabels
## pred_cross BCR/ABL NEG
##    BCR/ABL       5   3
##    NEG           0   3
```

```
Accuracy_cross = 100 * sum(testLabels == pred_cross)/length(testLabels)
Accuracy_cross
```

```
## [1] 72.72727
```

```
pred_boot <- knn(train = t(training), test = t(testing),
            cl = trainLabels, k=knn.boot$best.parameters[1,1])
table(pred_boot, testLabels)
```

```
##          testLabels
## pred_boot BCR/ABL NEG
##   BCR/ABL       5   2
##   NEG           0   4
```

```
Accuracy_boot = 100 * sum(testLabels == pred_boot)/length(testLabels)
Accuracy_boot
```

```
## [1] 81.81818
```

```
pred_fix <- knn(train = t(training), test = t(testing),
            cl = trainLabels, k=knn.fix$best.parameters[1,1])
table(pred_fix, testLabels)
```

```
##          testLabels
## pred_fix  BCR/ABL NEG
##   BCR/ABL       5   2
##   NEG           0   4
```

```
Accuracy_fix = 100 * sum(testLabels == pred_fix)/length(testLabels)
Accuracy_fix
```

```
## [1] 81.81818
```

```
barplot(c(Accuracy_cross,Accuracy_boot,Accuracy_fix),main="Accuracy",
   ylab="Percentage",ylim=c(0,80),col=c("darkblue", "black","red"))


legend("bottomright", legend =
         c("cross validation", "bootstraping","fixed set"),
         fill = c("darkblue", "black","red"))
```