# Solutions for chapter Supervised Machine Learning

## *Exercise 1*

```
> table(ALL_bcrneg$mol.biol)
BCR/ABL     NEG
     37      42
```

## *Exercise 2*

```
> class(ALLfilt_bcrneg)
[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"
```

## *Exercise 3*

The distances available include Kullback-Leibler distance, mutual information distance, Euclidean distance, Manhattan distance, and correlation distance (using Pearson, Spearman, or Kendall's tau). See the `dist` function and the `daisy` function in the **cluster** package for other distances.

## *Exercise 4*

The diagonal band of blue squares is probably the most prominent feature, but it is merely indicating that each sample is distance zero from itself. After that you might notice that there are some bluish colored blocks along the diagonal, the most prominent one being in the top-left corner. The dendrogram also suggests that those samples are similar to each other, and some distance from the others.

## *Exercise 5*

Because the **bioDist** package is loaded we can simply call the `spearman.dist` function. All other steps are essentially the same, as before.

```
> spD =  spearman.dist(ALLfilt_bcrneg)
> ## spD@Size
> attr(spD, "Size")
[1] 79
> spM = as.matrix(spD)
```

```
> heatmap(spM, sym=TRUE, col=hmcol,
      distfun=function(x) as.dist(x))
```

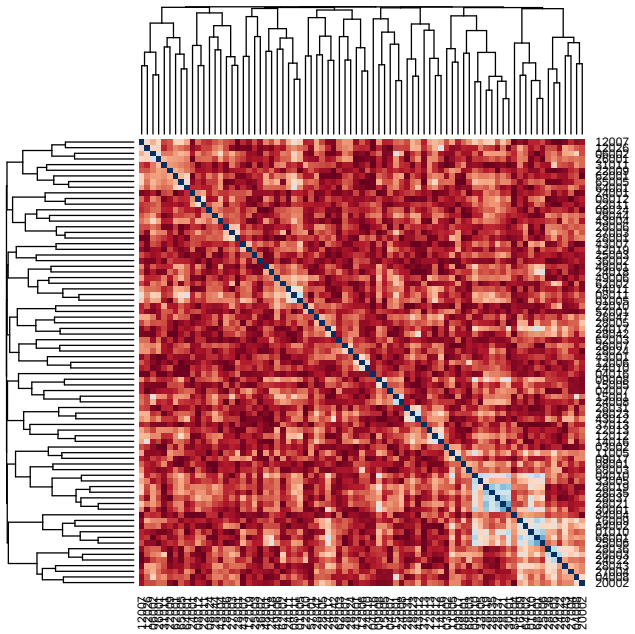

Figure 1. A heatmap of the between-sample distances, for the same data as in Figure **??**, but now using Spearman's correlation instead of the Euclidean distance.

In this heatmap, the samples seem to be farther from each other (darker red colors predominate), but there are a small number that are quite close, as evidenced by the light blue rectangle in the middle of the heatmap.

## *Exercise 6*

```
> cD = MIdist(ALLfilt_bcrneg)
> cM = as.matrix(cD)
> closest.top("03002", cM, 1)
[1] "09017"
```

## *Exercise 7*

We use the `MLearn` interface to the machine learning code. We make use of the MLearn interface to the different machine learning tools, provided by `MLearn`.

The function, `confuMat`, can be used to compute the confusion matrix, and from that we can estimate the error rates.

```
> kans = MLearn( mol.biol ~ ., data=ALLfilt_bcrneg,
      knnI(k=1,l=0), TrainInd)
> confuMat(kans)
         predicted
given     BCR/ABL NEG
  BCR/ABL      12   5
  NEG          10  12
> dldans = MLearn( mol.biol ~ ., ALLfilt_bcrneg, dldaI,
      TrainInd)
> confuMat(dldans)
         predicted
given     BCR/ABL NEG
  BCR/ABL      10   7
  NEG           8  14
```

```
> ldaans = MLearn( mol.biol ~ ., ALLfilt_bcrneg, ldaI,
      TrainInd)
> confuMat(ldaans)
         predicted
given     BCR/ABL NEG
  BCR/ABL      15   2
  NEG           7  15
```

## *Exercise 8*

a Ties are broken at random. This suggests that it might not be all that helpful to select a value of $k$ that is even, as different users would potentially classify samples differently, given the same data.

b This is difficult with the current implementation. You would essentially need to do the nearest neighbor finding directly from the distance matrix, and this will be somewhat slow. The `closest.top` function, from the **bioDist** package could be used.

c The `knn` function has a parameter `prob` that if set to `TRUE` will cause the proportion of votes for the winning class to be returned. This could be used. Also, the parameter `l` can be used; in that case doubt is encoded as NA. The concept of outlier is more difficult, but could potentially be handled in a preprocessing step. Any object that is a long way from all other objects could be identified as an outlier and removed. This does not help with pairs of outliers, or triples.

## *Exercise 9*

We repeat the steps taken above, but use all of the data.

```
> alltt = rowttests(ALLfilt_bcrneg, "mol.biol")
> ordall = order(abs(alltt$statistic), decreasing=TRUE)
> fNall = featureNames(ALLfilt_bcrneg)[ordall[1:50]]
> intersect(fNall, fNtt)
 [1] "1635_at"    "1674_at"    "40504_at"   "40202_at"
 [5] "32434_at"   "37027_at"   "40167_s_at" "37403_at"
 [9] "40480_s_at" "33774_at"   "36591_at"   "37363_at"
[13] "34472_at"   "35162_s_at" "37014_at"   "32542_at"
[17] "33362_at"   "33440_at"   "36275_at"   "40516_at"
[21] "1467_at"    "40076_at"   "106_at"     "38994_at"
[25] "1249_at"    "36638_at"
```

## *Exercise 10*

We simply redo the calls with

```
>  dldtt = MLearn( mol.biol ~ ., BNf, dldaI, TrainInd)
>  confuMat(dldtt)
         predicted
given     BCR/ABL NEG
  BCR/ABL      11   6
  NEG           3  19
>  ldatt  = MLearn( mol.biol ~ ., BNf, ldaI, TrainInd)
>  confuMat(ldatt)
         predicted
given     BCR/ABL NEG
  BCR/ABL      15   2
  NEG           3  19
```

In all cases the error rates are lower, which is nice.

## *Exercise 11*

Each sample is left out, in turn, and because $k = 1$ the class of that sample is determined by its nearest neighbor in the remaining $n - 1$ samples. For larger values of $k$, then more nearest neighbors would be used in the prediction. The confusion matrix is produced by the `confuMat` function, and it can be used to estimate either the overall error rate, or the class conditional error rates.

```
> knnCM = confuMat(knnXval1)
> knnCM
         predicted
given     BCR/ABL NEG
  BCR/ABL      32   5
  NEG          16  26
> #overall error rate
> (knnCM[1,2] + knnCM[2,1])/sum(knnCM)
[1] 0.266
> #class conditional error rates
> knnCM[1,2]/sum(knnCM[1,])
[1] 0.135
> knnCM[2,1]/sum(knnCM[2,])
[1] 0.381
```

So it seems that it was harder to predict the NEG phenotype than the BCR/ABL phenotype.

## *Exercise 12*

a
```
> lk3f2 = MLearn(mol.biol~., data=BNx, knnI(k=1),
        xvalSpec("LOO", fsFun=fs.absT(5)))
> confuMat(lk3f2)
> table(unlist(fsHistory(lk3f2)))
```

The error rate seems to be bit higher when only five features are selected.

## *Exercise 13*

This is quite an interesting problem. Basically, what you need to do is to try out the KNN algorithm, for a variety of values of $k$, and see what value of $k$ gives the lowest error rate.

```
> knnXval2 = MLearn(mol.biol~., data=BNx, knn.cvI(k=2, l=0),
      trainInd=1:ncol(BNx))
> confuMat(knnXval2)
```

```
> knnXval3 = MLearn(mol.biol~., data=BNx, knn.cvI(k=3, l=0),
      trainInd=1:ncol(BNx))
> confuMat(knnXval3)
```

```
> knnXval5 = MLearn(mol.biol~., data=BNx, knn.cvI(k=5, l=0),
      trainInd=1:ncol(BNx))
> confuMat(knnXval5)
```

## *Exercise 14*

We are only concerned with the errors for the test set because those for the training set are known to be overly optimistic. Error rates can be computed for either all predictions combined, or on a per class basis. It is often the case that error rates can be quite different for different classes, so we also compute the class conditional error rates. Note that the error rates are a bit worse for model 2, which had a much smaller value of `mtry`.

```
> cf1 = confuMat(rf1)
> overallErrM1 = (cf1[2,1] + cf1[1,2])/sum(cf1)
> overallErrM1
```

```
[1] 0.179
> perClass1 = c(cf1[1,2], cf1[2,1])/rowSums(cf1)
> perClass1
BCR/ABL     NEG
  0.176   0.182
```

And now for model 2.

```
> cf2 = confuMat(rf2)
> overallErrM2 = (cf2[2,1] + cf2[1,2])/sum(cf2)
> overallErrM2
[1] 0.205
> perClass2 = c(cf2[1,2], cf2[2,1])/rowSums(cf2)
> perClass2
BCR/ABL     NEG
  0.294   0.136
```

For KNN we had the following error rates:

```
> cfKNN = confuMat(knnf)
> (cfKNN[1,2] + cfKNN[2,1])/sum(cfKNN)
[1] 0.282
> #class conditional error rates
> cfKNN[1,2]/sum(cfKNN[1,])
[1] 0.353
> cfKNN[2,1]/sum(cfKNN[2,])
[1] 0.227
```

And in this case, it seems that KNN has the lower overall error rate, 0.282 compared to 0.179 for model 1 and 0.205 for model 2.

## *Exercise 15*

We can obtain the importance measure by calling the importance function.

```
> impvars = function(x, which="MeanDecreaseAccuracy", k=10) {
      v1 = order(importance(x)[,which], decreasing=TRUE)
      importance(x)[v1[1:k],]
  }
> ivm1 = impvars(rf1@RObject, k=20)
> ivm2 = impvars(rf2@RObject, k=20)
> intersect(row.names(ivm1) , row.names(ivm2))
[1] "X1467_at"  "X40196_at" "X1635_at"  "X40504_at"
[5] "X37027_at" "X1674_at"  "X40202_at" "X32696_at"
```

The other importance measure is called `MeanDecreaseGini`, and we leave that part of the problem to the reader.

## *Exercise 16*

Reversing the role of the test and training sets is quite simple, we use model 2.

```
> rfRev = MLearn( mol.biol~., data=ALLfilt_bcrneg,
      randomForestI, TestInd, ntree=2000, mtry=10,
      importance=TRUE)
> rfRev
MLInterfaces classification output container
The call was:
MLearn(formula = mol.biol ~ ., data = ALLfilt_bcrneg, .m
ethod = randomForestI,
    trainInd = TestInd, ntree = 2000, mtry = 10, importa
    nce = TRUE)
```

```
Predicted outcome distribution for test set:

BCR/ABL     NEG
     19      21
```

and for the confusion matrix

```
> cfR = confuMat(rfRev)
> cfR
         predicted
given     BCR/ABL NEG
  BCR/ABL      14   6
  NEG           5  15
> overallErr = (cfR[2,1] + cfR[1,2])/sum(cfR)
> overallErr
[1] 0.275
> perClass = c(cfR[1,2], cfR[2,1])/rowSums(cfR)
> perClass
BCR/ABL     NEG
   0.30    0.25
```

We can see from the confusion matrix that the error rate observed is roughly comparable to that obtained with the other split, as we expected, inasmuch as the two sets were roughly the same size.

It is also quite simple to use the whole dataset to fit a random forest.

```
> rfAll = MLearn( mol.biol~., data=ALLfilt_bcrneg,
      randomForestI, 1:79, ntree=1000, mtry=10,
      importance=TRUE)
> rfAll@RObject
Call:
 randomForest(formula = formula, data = trdata, ntree =
 1000,      mtry = 10, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 10

        OOB estimate of  error rate: 19%
Confusion matrix:
        BCR/ABL NEG class.error
BCR/ABL      27  10       0.270
NEG           5  37       0.119
```

## *Exercise 17*

Using KNN is quite straightforward. We demonstrate its use for $k = 1$; you might want to try other methods.

```
> knn1MV = knn(t(exprs(trainSet)), t(exprs(testSet)),
      trainSet$mol.biol)
> tab1 = table(knn1MV, testSet$mol.biol)
> tab1
knn1MV     ALL1/AF4 BCR/ABL NEG
  ALL1/AF4        5       0   0
  BCR/ABL         0      15  13
  NEG             0       3   8
```

```
> s3 = table(testSet$mol.biol)
```

Class conditional error rates are estimated by considering those with the correct classification (on the diagonal of the table produced above). For example, of the 18 BCR/ABL samples in the test set, 15 are correctly classified, so that the class conditional error rate is 0.17.

It is not so easy to handle unbalanced data. One can, in principle, find $k$ nearest neighbors, and then compare the proportion of nearest neighbors to the class counts.