clst demo

Noah Hoffman

October 26, 2021

Contents

1 Introduction

clst performs supervised classification using a modified nearest-neighbor approach. This vignette demonstrates its use.

1.1 Definition of symbols

Consider a set of N objects that can be grouped into K classes with class labels $c_m, m \in 1...K$. Each object may be compared to another to generate a pairwise distance $d_{i,j}$. Additional symbol definitions are as follows:

d^w	all within-class pairwise distances.
d^b	all between-class pairwise distances.
d_m^w	within-group distances among members of class m.
$d_{x,i\in 1...N}$	distances between object x and each object in the reference set.
$d_{x,i\in c_m}$	distances between query object x and each object belonging to class c_m.
D	A value defining an optimal separation between within-class comparisons and between-class comparisons. In general, therefore, we would expect $d^w < D < d^b$.

Given these definitions, we can further define a match score s to describe the confidence that object x is a member of class c_m as follows:

$$s_{x,m} = \frac{d_{x,i\in c_m} < D}{d_{x,i\in 1...N} + d}$$

We include a small offset d in the denominator to prevent over-weighting match scores for small groups. Thus if $d = 0.5$, a query object with all $d_{x,m} < D$ will have a match score of $1/1 + 0.5 \approx 0.66$ for a class with a single member.

We can define an additional parameter C as the value of s_{x,c_m} above which object x is classified as a member of class c_m. (Note: C is defined by the `minScore` argument to the function `classify`.

1.2 Classification outcomes

Classification of an object x can be performed by considering the values of $s_{x,m}$ for all classes represented in a reference set. Possible outcomes of classification might include an unequivocal *match* with a single class; *confusion* between two or more possible classes; *no match*, which reflects confidence that the object can *not* be placed into any of the represented classes; or evidence that the object is a clear *outlier*. Given the simple case of two classes A and B and four query objects w, x, y, and z (illustrated in Figure 1, we might define these outcomes as follows:
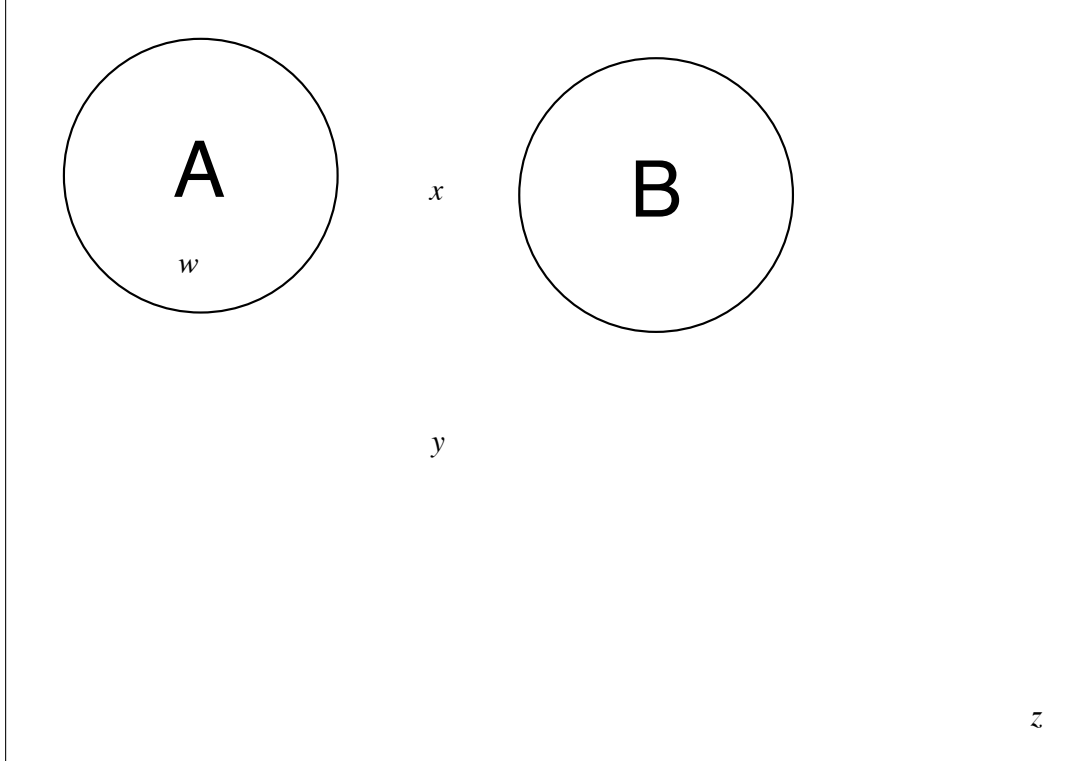

Figure 1: Members of classes A and B lie within respectively labeled circles; points w, x, y, and z indicate unclassified objects.

match (point w) $\begin{cases} s_{w,A} > C \\ s_{w,B} < C \end{cases}$

confusion (point x) $\begin{cases} s_{x,A} > C \\ s_{x,B} > C \end{cases}$

no match (point y) $\begin{cases} s_{y,A} < C \\ s_{y,B} < C \\ d_{y,i\in 1...N} < some\ d^b \end{cases}$

outlier (point z) $\begin{cases} s_{z,A} < C \\ s_{z,B} < C \\ d_{y,i\in 1...N} > most\ d^b \end{cases}$

Described in the context of the function `classify`, classification is performed as follows. The arguments `min-Score` and `doffset` provide the parameters C and d, respectively. On each iteration of the classification algorithm, pairwise distances in the reference set are defined as either within- or between-group based on the labels in `groups`. A threshold partitioning these two categories is determined as the point of maximal mutual information. If classification results in confusion between two or more categories, pairwise distances among objects in these remaining categories will be re-partitioned, and a new threshold determined. The `maxDepth` parameter defines the maximum number of times this re-partitioning may occur.

2 Iris example

Create a distance matrix (`dmat`) using the Iris data set, which describes phenotypic characteristics of three flower species. We also define a factor (`groups`) that associates each of the samples in the data set with one of three Iris species.

```
> library(clst)
> data(iris)
> dmat <- as.matrix(dist(iris[,1:4], method="euclidean"))
> groups <- iris$Species
```

The relationship among objects in the Iris data set can be illustrated using multidimensional scaling. The function `scaleDistPlot` creates an annotated scatterplot of the output of `cmdscale`.

We can calculate D as the value that results in maximal mutual information between the vector classifying distances as "within" or "between" and the vector indicating if the corresponding pairwise distance is greater than or less than D (method="mutinfo"). This is the default method used by the classifier.

```
> thresh <- findThreshold(dmat, groups, type="mutinfo")
> str(thresh)

List of 7
 $ D        : num 2.04
 $ pmmi     : num 0.395
 $ interval : num [1:2] 0.837 3.401
 $ breaks   :'data.frame':       259 obs. of  2 variables:
  ..$ x: num [1:259] 0.837 0.84 0.85 0.86 0.87 ...
  ..$ y: num [1:259] 0.238 0.24 0.243 0.246 0.25 ...
 $ distances:'data.frame':       11175 obs. of  4 variables:
  ..$ vals      : num [1:11175] 0.539 0.51 0.648 0.141 0.616 ...
  ..$ comparison: Ord.factor w/ 2 levels "within"<"between": 1 1 1 1 1 1 1 1 1 1 ...
  ..$ row       : int [1:11175] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ col       : int [1:11175] 2 3 4 5 6 7 8 9 10 11 ...
 $ method   : chr "mutinfo"
 $ params   :List of 5
  ..$ prob      : num 0.5
  ..$ roundCuts : num 2
  ..$ minCuts   : num 20
  ..$ maxCuts   : num 300
  ..$ targetCuts: num 100
```

By default, D has a lower and upper bound defined as the minimum of the between-class distances and maximum of the within-class distances. One can also set the upper and lower bounds of D as some quantile of the within class distances and between-class differences, respectively using the `prob` argument (the default is 0.5); the constraint can be removed by setting `prob=NA`

```
> thresh2 <- findThreshold(dmat, groups, type="mutinfo", prob=NA)
> print(thresh2$interval)

[1] 0.2236068 3.8236109
```

In this example, setting alternative bounds for D does not alter the results.

The plot method for the thresh object illustrates the underlying calculations performed using either method

```
> plot(do.call(plotDistances, thresh))
```

```
> ii <- c(1,125)
> plot(scaleDistPlot(dmat, groups, indices=ii,O=ii))
```
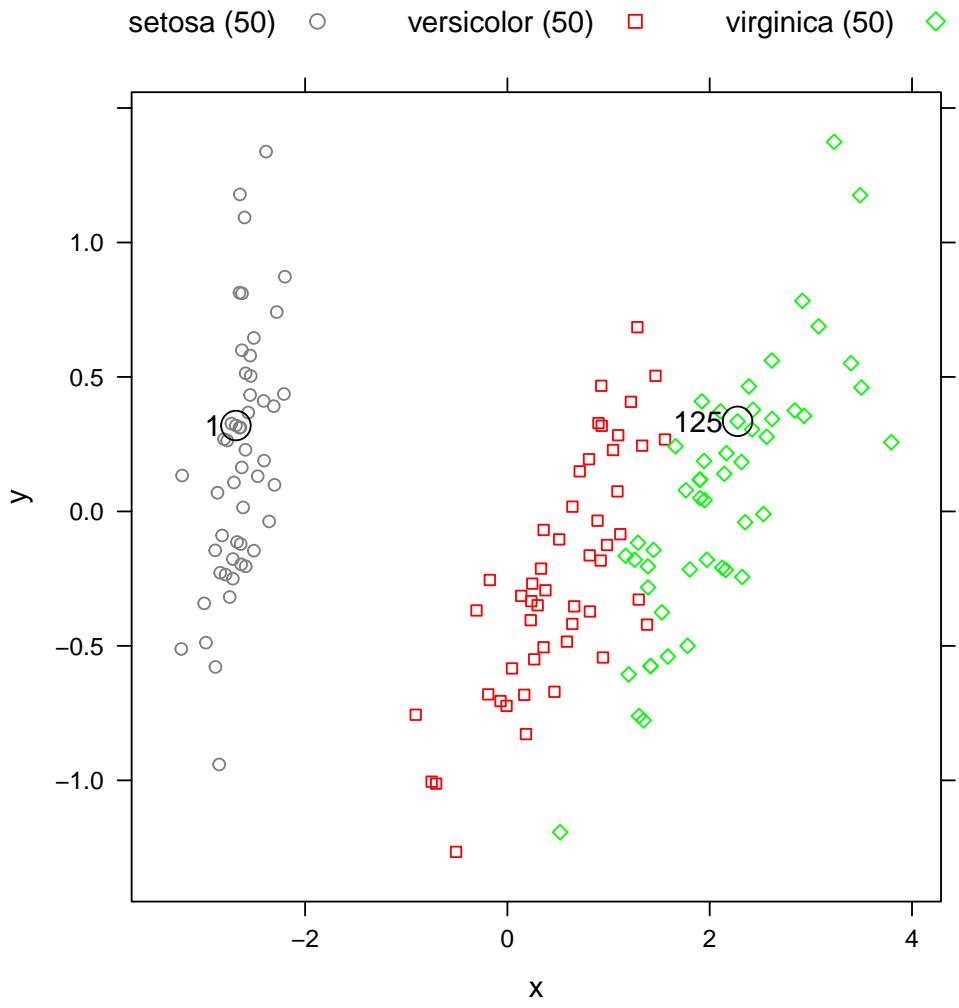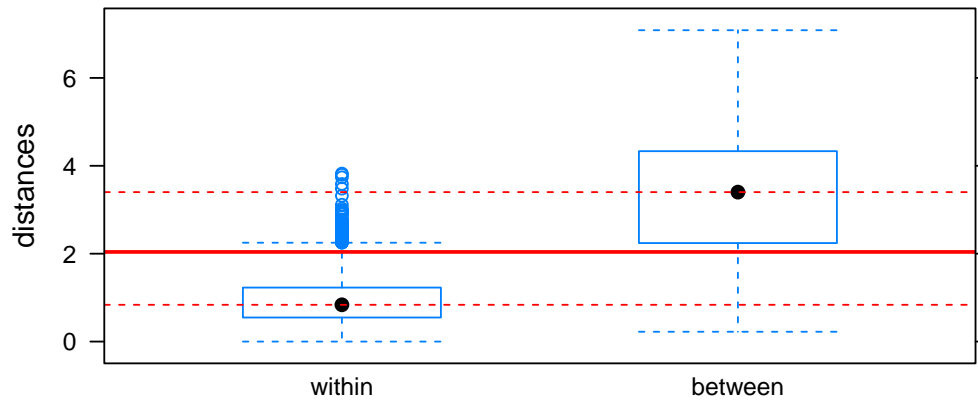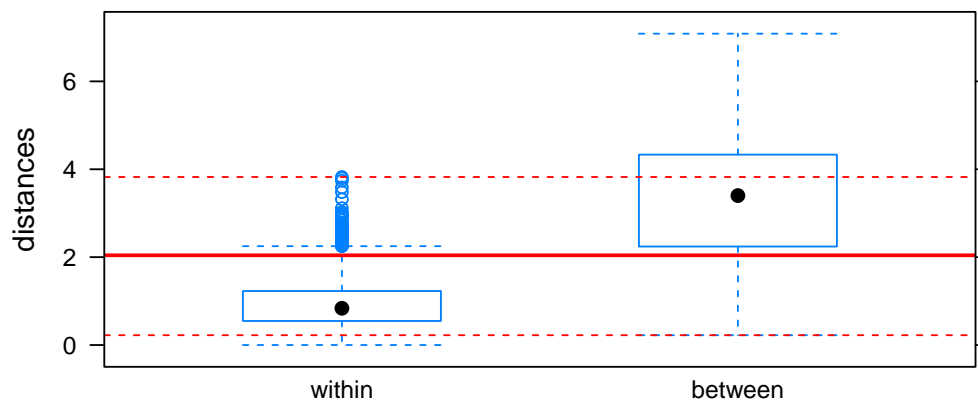

Figure 2: Visualization of the Iris data set using multidimensional scaling. Objects to be classified in the examples below are indicated.

```
> plot(do.call(plotDistances, thresh2))
```


This data set contains two classes (that is, flower species) that are closely related, and one more distantly related to the other two. Therefore the performance of a single D for predicting inter- versus intra-class distances is rather poor.

2.1 Classification of selected items

The function `classify` implements the classification algorithm. Inputs to the function include a square matrix of distances and a factor of class labels (`dmat` and `groups`) as well as `dvect`, a vector containing distances between the sample to be classified and each of the reference objects.

The classification algorithm requires the two parameters, C and d, which are provided to `classify` as arguments `minScore` and `doffset`.

The more important of the two parameters, `minScore`, defines the score cutoff for group membership; that is, if an unknown sample has a proportion of similarity scores > `minScore` when compared to reference samples in a given class, it is considered a match for that class. We will experiment with the effect of changing the value of `minScore` later.

`doffset` is used in the calculation of the match score; its purpose is to reduce the weight of scores resulting from matches to members of a group with very few members. Thus with the default value of `doffset`=0.5, if a query object has a distance $< D$ to an object in a group of size 1, the resulting score is $1/(1 + 0.5) = 0.66$.

To provide an artificial example, we can remove a single sample from the Iris data and perform classification using the remaining samples as a reference set.

```
> ind <- 1
> species <- gettextf('I. %s', groups[ind])
> cat('class of "unknown" sample is',species)

class of "unknown" sample is I. setosa

> dmat1 <- dmat[-ind,-ind]
> groups1 <- groups[-ind]
> dvect1 <- dmat[ind, -ind]
> cc <- classify(dmat1, groups1, dvect1)
> printClst(cc)

================================================================
 matches: setosa
================================================================


--------------- rank  (depth 1) --------------------
thresh: prob = 0.5
classifer: minScore = 0.45  doffset = 0.5 dStart = NA
method = mutinfo  D = 2.04 pmmi = 0.391
matches: setosa
           below above score match  min median  max
setosa        49     0  0.99      1 0.10   0.51 1.35
versicolor     0    50  0.00      0 2.09   3.38 4.24
virginica      0    50  0.00      0 3.59   4.74 6.50
-------------------------------------------------
```

The query sequence is a member of the species *I. setosa*, which is relatively divergent from the other two, so our classifier performs well on the first try.

Classification of members of the other two Iris species present a more challenging case:

```
> ind <- 125
> species = gettextf('I. %s', groups[ind])
> pp <- pull(dmat, groups, ind)
> cc <- do.call(classify, pp)
> cat(paste('class of "unknown" sample is', species))

class of "unknown" sample is I. virginica

> printClst(cc)

================================================================
 matches: virginica
================================================================


--------------- rank  (depth 1) --------------------
thresh: prob = 0.5
classifer: minScore = 0.45  doffset = 0.5 dStart = NA
method = mutinfo  D = 2.04 pmmi = 0.396
matches: versicolor, virginica
           below above score match  min median  max
virginica     48     1  0.97      1 0.30   0.81 2.34
versicolor    30    20  0.59      1 0.86   1.85 3.39
setosa         0    50  0.00      0 4.49   4.94 5.57

--------------- rank  (depth 2) --------------------
thresh: prob = 0.5
classifer: minScore = 0.45  doffset = 0.5 dStart = NA
```

```
method = mutinfo  D = 1.09 pmmi = 0.122
matches: virginica
          below above score match  min median  max
virginica    31    18  0.63     1 0.30   0.81 2.34
versicolor    2    48  0.04     0 0.86   1.85 3.39
-------------------------------------------------
```

2.2 Leave-one-out analysis

The performance of the classifier for a data set using a given set of parameters can be examined using a leave-one-out analysis. The list `loo` contains the results of the final iteration of classification for each object in the iris data set.

```
> loo <- lapply(seq_along(groups), function(i){
+   do.call(classify, pull(dmat, groups, i))
+ })
> matches <- lapply(loo, function(x) rev(x)[[1]]$matches)
> result <- sapply(matches, paste, collapse='-')
> table(ifelse(result=='','no match',result),groups)

                      groups
                       setosa versicolor virginica
  setosa                   50          0         0
  setosa-versicolor         0          1         0
  versicolor                0         43         0
  versicolor-virginica      0          5         9
  virginica                 0          1        41
```

Note that there is some confusion between versicolor and virginica. The objects that could not be assigned to a single taxon are indicated in Figure 3.

The specificity of the classifier can be improved by increasing the value of `minScore`, at the cost of a decrease in sensitivity (Figure 4).

```
> loo <- lapply(seq_along(groups), function(i){
+   do.call(classify, c(pull(dmat, groups, i),minScore=0.65))
+ })
> matches <- lapply(loo, function(x) rev(x)[[1]]$matches)
> result <- sapply(matches, paste, collapse='-')
> table(ifelse(result=='','no match',result),groups)

                      groups
                       setosa versicolor virginica
  no match                  0          0         2
  setosa                   50          0         0
  versicolor                0         38         4
  versicolor-virginica      0         11        13
  virginica                 0          1        31
```

Finally, we can see a summary of the classification results of an item for which the result was "no match."

```
> printClst(loo[[118]])

=============================================================
 matches: (no match)
=============================================================


--------------- rank  (depth 1) --------------------
thresh: prob = 0.5
classifer: minScore = 0.65  doffset = 0.5 dStart = NA
```

```
> confusion <- sapply(matches, length) > 1
> no_match <- sapply(matches, length) < 1
> plot(scaleDistPlot(dmat, groups, fill=confusion, O=confusion, X=no_match))
```
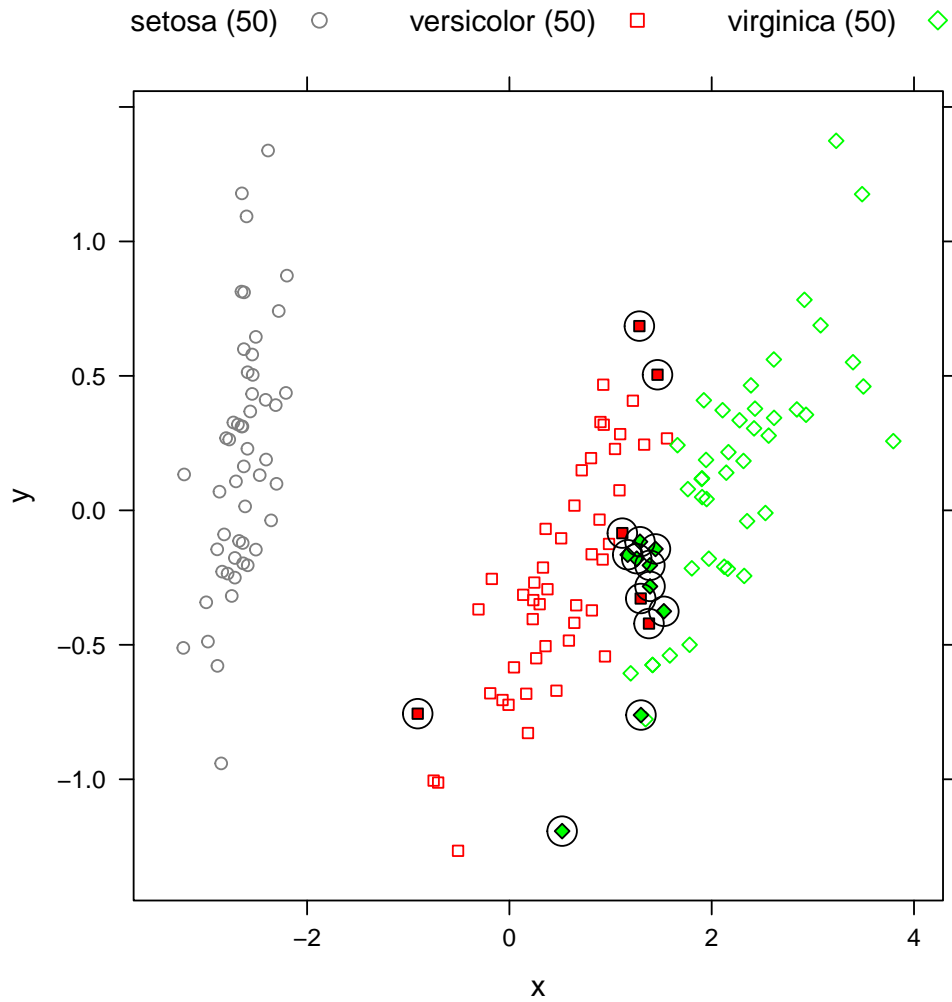

Figure 3: Visualization of the Iris data set using multidimensional scaling. Objects with a classification output of confusion are filled and indicated with a circle.

```
>  confusion <- sapply(matches, length) > 1
>  no_match <- sapply(matches, length) < 1
>  plot(scaleDistPlot(dmat, groups, fill=confusion, O=confusion,
+                      X=no_match, indices=no_match))
```
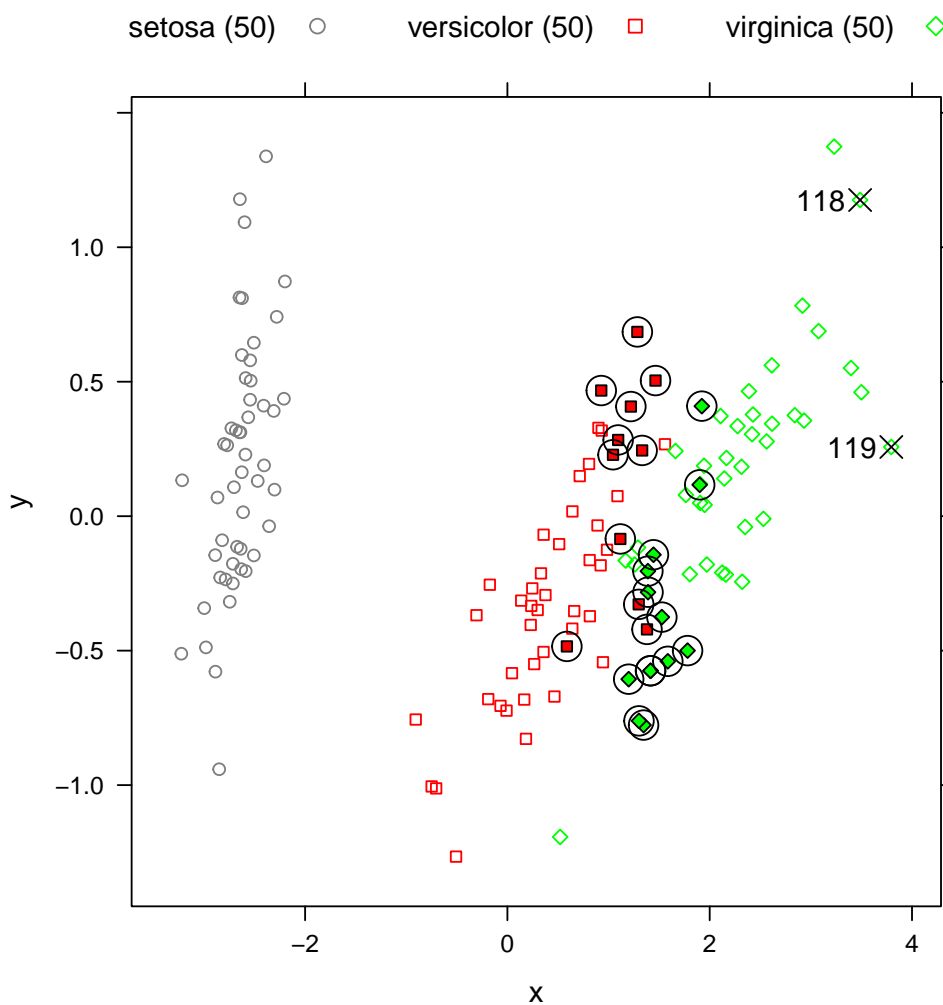

Figure 4: Visualization of the Iris data set using multidimensional scaling. Objects with a classification output of confusion are filled and indicated with a circle. Unclassified items are crossed out. Here `minScore=0.65`

```
method = mutinfo  D = 2.04 pmmi = 0.398
matches:
          below above score match  min median  max
virginica    31    18  0.63      0 0.41   1.94 3.82
versicolor    0    50  0.00      0 2.19   3.26 4.83
setosa        0    50  0.00      0 5.71   6.21 6.93
-----------------------------------------------
```