Advanced use of **FIEmspro**

David P. Enot

October 16, 2007

```
randomForest 4.5-18
Type rfNews() to see new features/changes/bug fixes.
```

1 Introduction

2 Functionalities of `accest`

`accest` is a wrapper function for calculating classification estimates using predefined data partitioning sets. As for `accest`, this function loop over each set of data, generate generate training and test data and then evaluate predictions of a classifier built on the training data. Unless the classifier can only cope with two-class problems, these functions allow the manipulation of any problem complexity. Three types of estimates are given for each replication: accuracy, so-called margin and AUC are calculated by `accest`. The fundamental difference between the two functions is that `accest` requires the methods that must be explictly defined whereas `accest` requires generic classifiers that fulfil R standards to define predictive techniques such as the ones available in packages like MASS or e1071. In the current implementation, several differences do exist. As input:

- `accest` accepts datasets in a form of a list containing X and Y matrices and optionally validation parameters in addition to the class data matrix/class vector couple and formula type.
- To speed up calculations, `accest` can be run on parallel nodes (i.e. multicore processors or workstation network)
- `accest` allows a better control of the random number generator not only for issues related for parallelisation but also for reproducibility purposes with algorithms such randomForest
- Control if iterations should be printed or not

The output of `accest` is the same as for `accest` except that few features have been added:

- Arguments used by the classifier can be useful to keep trace of the classifier parametrisation (for e.g one might compare RF using ntree=10 and RF using ntree=1000).

- Extra information provided by the user defined classifier are also kept so that any important piece of information computed during the training phase can be extracted with little efforts.
- Class predictions and decision boundary information are kept for post analysis of the classifier properties or comparison of the classifier abilities.
- Classification task is available to keep trace of the classification problem.

3 Classifiers design

Despite the availability of 10s of different classifier techniques in R, the main drawback is that only very few fulfil R standard requirement to call and output the outcome of of `accest` is that the classification method must be implemented. However, this effort is quickly rewarded by numerous advantages regarding the flexibility to create new classifier and the possibility to integrate code that do not fit `accest` constraints. To illustrate this point, the two functions below are simply reproducing randomForest discrimination algorithm:

```
sameasrf <- function(data,...){

  ## Possible arguments
  dots <- list(...)

  ## Build RF model and predict data$te
  mod  <- randomForest(data$tr,data$cl,...)
  ## Get votes on data$te
  prob <- predict(mod,data$te,type="vote")
  ## Get hard predictions
  pred <- predict(mod,data$te)

  ## Return a list
  ## here mod does not contain anything
  list(mod=NULL,pred=pred,prob=prob,arg=dots)
}
```

All what a user defined function needs as input is a list `data` that may contain:

- tr: training data X matrix
- cl: training example labels
- te: test data X matrix
- clte: training example labels
- ltr: indices of the training examples in the original dataset

Optionally, further arguments (...) can be passed or not to the classifier (here optional arguments will be passed to the core randomForest function). The output of such classifier is a list of four elements:

- mod - Anything related to the classifier. The content of mod is kept for post processing analysis of user defined relevant properties.
- pred - A factor vector containing the prediction made from dat.te.
- prob - A table where posteriori probabilities or equivalent are transferred in order to calculate decision boundary and margins in `accest`.
- arg - simply contains a list of arguments.

4 Simple example

This simple is given to show the equivalence between running `accest` with different calls. The classification task consists in discriminating the famous 3 iris species. For more fun, a random selection of 30 sample labels have been permuted:

```
>data(iris)
>dat = as.matrix(iris[, 1:4])
>cl = iris[, 5]
>lrnd = sample(1:150)[1:30]
>cl[lrnd] = sample(cl[lrnd])
```

We first define the data partitioning mechanism. In this case, we have adopted a 10 bootstrap resampling strategy, repeated twice:

```
>pars <- valipars(sampling = "boot", niter = 2,
+     nreps = 10)
>tr.idx <- trainind(iris$Species, pars)
```

accest and `accest` are run in the same conditions and results can be compared:

```
>ntree = 1000
>set.seed(70)
>accrf <- accest(dat, cl, clmeth = "randomForest",
+     pars = pars, tr.idx = tr.idx, ntree = ntree)

ACCEST.1 Iteration (2): 1 2

>set.seed(70)
>acc.1rf <- accest(dat, cl, clmeth = "sameasrf",
+     pars = pars, tr.idx = tr.idx, ntree = ntree)

ACCEST.1 Iteration (2): 1 2

>summary(accrf)

Method:                  randomForest
Arguments:          ntree=1000
Discrimination:          setosa~versicolor~virginica

No. of iteration:          2
```

```
Sampling:                 bootstrap
No. of replications:         10

Accuracy:                 0.823
Margin:                         0.499

Overall confusion matrix of training data:
            all.pred
all.cl       setosa versicolor virginica
  setosa        325         31        13
  versicolor     22        289        44
  virginica      15         74       310

Accuracy on each iteration:
    1     2
0.826 0.819
```

>summary(acc.1rf)

```
Method:                 sameasrf
Arguments:         ntree=1000
Discrimination:         setosa~versicolor~virginica

No. of iteration:         2
Sampling:                 bootstrap
No. of replications:         10

Accuracy:                 0.823
Margin:                         0.499

Overall confusion matrix of training data:
            all.pred
all.cl       setosa versicolor virginica
  setosa        325         31        13
  versicolor     22        289        44
  virginica      15         74       310

Accuracy on each iteration:
    1     2
0.826 0.819
```

5 Simple example for comparing classifiers

An interesting aspect of customising your classifier lies on the fact that complex task can implemented easily. One of these is the need of proper double cross validation to for e.g. determine the number of components to use in a PLS model. Another issue relates to proper validation of classifiers with an embedded feature selection step where the selection of relevant features must be done on each individual training set before evaluating classifier predictive abilities using the reduced data. The following function `featselrf` illustrates this point: we want to test the fact that selecting features with AUC greater than a defined threshold(thres improves Random Forest classification. During the validation process, we are also interested in knowing which variables have selected as well as the average margin of the classifier:

```
featselrf <- function(data,thres=0.8,...){

  dots <- list(...,thres=thres)

  ## Select the best variables according to the univariate AUC
  fs.res=fs.auc(data$tr,data$cl)
  listvar<-which(fs.res$stats>=thres)

  ## Quick fix for case where no feature are selected
  ## by taking the best one !
  if(length(listvar)==0){listvar=which(fs.res$fs.rank==1)}

  ## Build and predict using listvar
  mod=randomForest(data$tr[,listvar,drop=FALSE],cl.tr,...)
  mod.mar=mean(FIEmspro:::marg(mod$votes,data$cl),na.rm=TRUE)
  prob=predict(mod,data$te[,listvar,drop=FALSE],type="vote")
  pred=predict(mod,data$te[,listvar,drop=FALSE])

  ## For illustration, mod contains  some elements that can be processed
  ##  - the list of variables used in the modelling
  ##  - the average training data margin
  list(mod=list(vkeep=listvar,mar=mod.mar),
     pred=pred,prob=prob,arg=dots)

}
```

Simple example involving classes ”1” and ”H” in the positive mode of Abr1. A 10- cross validation repeated 5 times resampling strategy is adopted and the partitioning of the data (variable tridx) is kept to allow strict comparison of the models:

```
>data(abr1)
>dat <- as.matrix(abr1$pos)
>y <- factor(abr1$fact$day)
>l = which(y == "1" | y == "H")
>y = factor(y[l])
>x <- preproc(dat[l, ], y = y, method = c("log",
+     "TICnorm"), add = 10)[, 110:1000]
>pars <- valipars(sampling = "cv", niter = 5, nreps = 10)
>tridx = trainind(y, pars)
```

Let’s play with the first comparison

```
>ntree = 1000
>acc.ori <- accest(x, y, clmeth = "randomForest",
+     ntree = ntree, pars = pars, tr.idx = tridx)
```

```
>acc.fs <- accest(x, y, clmeth = "featselrf", thres = 0.8,
+     ntree = ntree, pars = pars, tr.idx = tridx)
```

The following couple of command lines illustrate the benefits of keeping some relevant information in the field `mod` of the designed functions. Initially, one may be interested to know the actual classifier margin when calculated on the reduced data set. For this, `parse_vec` can be employed to retrieve a single value (or a vector of fixed length) contained in the `mod` component `acc.fs` `accest` result list

```
>mod.mar <- parse_vec(acc.fs$mod, "mar")
>print(mod.mar)
```

```
              1         2         3         4         5
 [1,] 0.6048592 0.5891361 0.5512730 0.5718900 0.5998948
 [2,] 0.5810958 0.5501153 0.5642818 0.5671209 0.5725256
 [3,] 0.5726227 0.5656413 0.6044914 0.6006592 0.5632466
 [4,] 0.6327399 0.5544643 0.5418408 0.5573158 0.6018214
 [5,] 0.5828614 0.5686539 0.5597689 0.5563240 0.5645886
 [6,] 0.5735325 0.5705638 0.5927953 0.5910292 0.5554221
 [7,] 0.5901023 0.6084433 0.6115622 0.5595218 0.6087871
 [8,] 0.5826032 0.5919817 0.6043900 0.5711402 0.5598140
 [9,] 0.5858241 0.5839319 0.5384233 0.5846734 0.5988981
[10,] 0.5810726 0.5813648 0.5948212 0.5974576 0.5565579
```

Another function has been designed to, this time, retrieve the frequency of the members of a list contained in the `mod` component of the `accest`. In the illustrated example, we only print out the 5 most frequent variables selected by AUC:

```
>var.freq <- parse_freq(acc.fs$mod, "vkeep")
>print(var.freq[1:5, ])
```

```
     lvar lfreq
P141   32    50
P236  127    50
P257  148    50
P274  165    50
P371  262    50
```

The following lines consist in aggregating the results into one list. In the scope of this example is to ease the output and comparison of the results. For visualisation purposes, names in two columns of `lclas` have been modified, the rest have been generated automatically. Therefore, for traceability purposes, it is advised that these first 6 columns should not be modified.

```
>lclas = mc.agg(acc.ori, acc.fs)
>lclas
```

```
  Model   Alg            Arg
1 "Mod_1" "randomForest" "ntree=1000"
2 "Mod_2" "featselrf"    "thres=0.8,ntree=1000"
```

```
  Pars          Dis   AlgId   DisId   Other
1 "5xNSt-10-cv" "1~H" "Alg_1" "Dis_1" ""
2 "5xNSt-10-cv" "1~H" "Alg_2" "Dis_1" ""
```

```
>lclas$cldef[, 7] = lclas$cldef[, 5]
>lclas$cldef[, 6] = c("RF", "AUC+RF")
>lclas
```

```
  Model   Alg            Arg
1 "Mod_1" "randomForest" "ntree=1000"
2 "Mod_2" "featselrf"    "thres=0.8,ntree=1000"
  Pars          Dis   AlgId    DisId Other
1 "5xNSt-10-cv" "1~H" "RF"     "1~H" ""
2 "5xNSt-10-cv" "1~H" "AUC+RF" "1~H" ""
```

The aims of the following section are to get a greater insight into the real differences between the two classification procedures. As such, `mc.summary` is a convenience function to output statistics related to accuracy, AUC and margins for a selection of models. It is simply called by:

```
>mcomp.sum <- mc.summary(lclas)
>mcomp.sum
```

```
 Multiple classifiers predictions summary :

 1~H
       Acc.mean Acc.sd Mar.mean Mar.sd AUC.mean AUC.sd
RF        0.890  0.029    0.184  0.015    0.933  0.031
AUC+RF    0.915  0.014    0.476  0.015    0.950  0.034
```

`mc.comp.1` tests for significant differences between predictions made by the two classifiers:

```
>mcomp.res <- mc.comp.1(lclas, p.adjust.method = "holm")
>mcomp.res
```

```
 Multiple classifiers comparisons :

 1~H
           Diff   Var t-stat DF  pval  holm
AUC+RF-RF 0.025 0.018  0.508 49 0.614 0.614
```

Aggregated ROC curves over each iteration and fold can be directly investigated and plotted

```
>mcomp.auc = mc.roc(lclas, 1:2, method = "thres")
>plot(mcomp.auc, leg = "AlgId")
```

Predictive margins for both classifiers are also simply compared:

```
>mar.iter = mc.meas.iter(lclas, type = "mar", nam = "AlgId")
>boxplot(mar.iter)
```

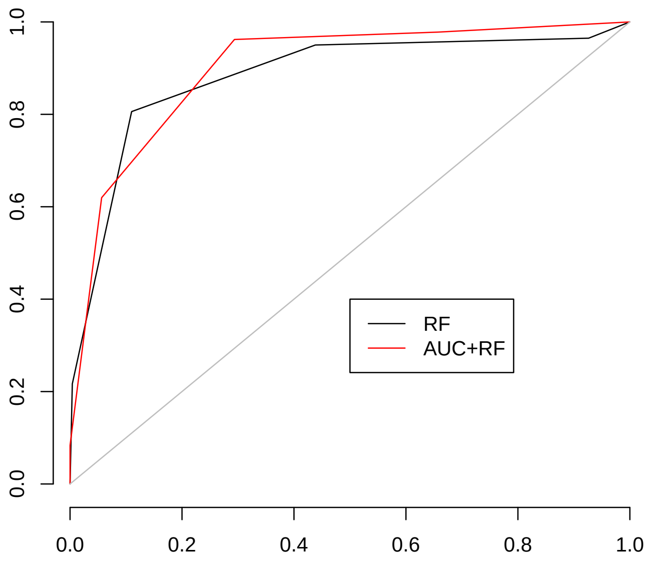

Figure 1: Aggregated FPR and TPR for the two classifiers

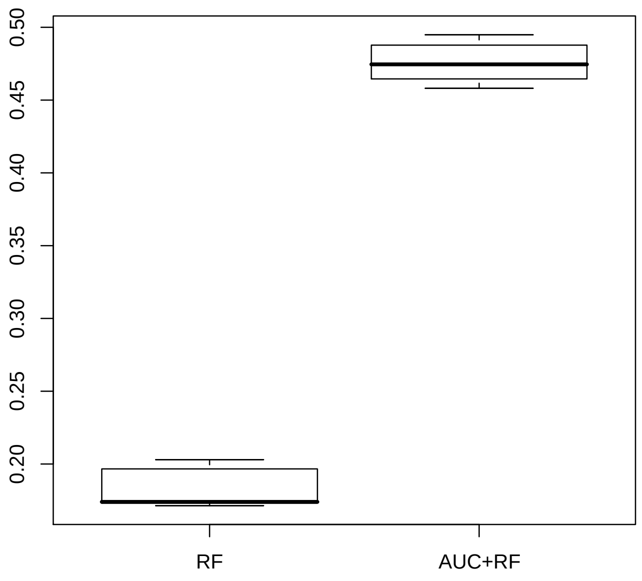

Figure 2: Distributions of the average predictive margins in the two classifiers

A simple t.test can then be optionally run to test for mean equality of margins in the two models:

```
>t.test(mar.iter[, 1], mar.iter[, 2], paired = TRUE)

        Paired t-test

data:  mar.iter[, 1] and mar.iter[, 2]
t = -117.187, df = 4, p-value = 3.18e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2992458 -0.2853942
sample estimates:
mean of the differences
               -0.29232
```

6 Multiple classifiers and/or multiple discrimination problems

Analysis of multiple classifiers performed on several discrimination problems can become tricky in a sense that it requires a lot of programming to handle similar tasks on multiple datasets. The aim of this secion is to illustrate an advanced use of the package to handle such problem in the most effective way.

6.1 List of datasets

One way to alleviate generation of several variables containing data matrices, class information or validation strategy. Each subset can enter `accest` and `feat.rank.re` analysis without specifying both class and data matrix. Additionally, each subset can also contain validation parameters so that direct comparison between classifiers and feature ranking technique can be easily done. `dat.sel1` is a that function allows generation of a selection of pairwise problems and/or multiple class problems. For e.g, several pairwise comparisons are selected and an identical data partitioning is performed on each data subset:

```
>data(abr1)
>dat <- abr1$pos
>y <- factor(abr1$fact$day)
>x <- preproc(dat, y = y, method = c("log", "TICnorm"),
+     add = 10)[, 110:1000]
>pars <- valipars(sampling = "cv", niter = 5, nreps = 10)
>lpwise = list(c("1", "H"), c("1", "2"), c("2",
+     "3"), c("3", "4"), c("4", "5"))
>dat1 = dat.sel1(x, y, pwise = lpwise, mclass = NULL,
+     pars = pars)
```

Print out the comparisons available in `dat1`:

```
>unlist(lapply(dat1, function(x) x$name))

[1] "1~H" "1~2" "2~3" "3~4" "4~5"
```

6.2 Multiple classifier comparisons in a few lines

Function `lapply` is a useful command to apply a function to a list of elements in a list. Here the list is the list of data subsets and the function is calling accest with different parameters. In the following, Random Forest, nlda and SVM are applied consecutively to each subsets. The main reason why each classification task are kept apart comes from the fact that either two different algorithms may share conflictual arguments (e.g. argument `scale` in `nlda` and `svm`) or that the same discrimination method may be run with two different parametrisations:

```
>resrf = lapply(dat1, function(x) accest(x, clmeth = "randomForest",
+     ntree = 1000))
>resnlda = lapply(dat1, function(x) accest(x, clmeth = "nlda"))
>ressvm = lapply(dat1, function(x) accest(x, clmeth = "svm",
+     kernel = "linear", cost = 0.1))
```

Overall, 3x5 `accest` have been formed. To simplify the handling of these classifiers, `mc.agg` is a routine that can be applied to aggregate then all in one list. The other advantage is that, in this case, only two variables are saved for later processing.

```
>lclas <- mc.agg(resrf, resnlda, ressvm)
>save(lclas, dat1, file = "mcexample.rda")
```

As before, it is also simpler to compare each classifier performance with `mc.comp.1`:

```
>lclas$cldef[, 7] <- lclas$cldef[, 5]
>mcomp.res <- mc.comp.1(lclas, p.adjust.method = "holm")
>mcomp.res
```

```
 Multiple classifiers comparisons :

 1~H
             Diff   Var t-stat DF  pval holm
Alg_2-Alg_1 -0.020 0.015 -0.453 49 0.653    1
Alg_3-Alg_1  0.005 0.014  0.117 49 0.908    1
Alg_3-Alg_2  0.025 0.013  0.597 49 0.554    1

 1~2
            Diff Var t-stat DF pval holm
Alg_2-Alg_1    0   0      0 49    1    1
Alg_3-Alg_1    0   0      0 49    1    1
Alg_3-Alg_2    0   0      0 49    1    1

 2~3
             Diff   Var t-stat DF  pval  holm
Alg_2-Alg_1 -0.035 0.010 -0.956 49 0.344 0.688
Alg_3-Alg_1  0.020 0.005  0.806 49 0.424 0.688
Alg_3-Alg_2  0.055 0.011  1.452 49 0.153 0.459

 3~4
```

```
             Diff   Var t-stat DF  pval holm
Alg_2-Alg_1 -0.01 0.002 -0.558 49 0.579    1
Alg_3-Alg_1  0.00 0.000  0.000 49 1.000    1
Alg_3-Alg_2  0.01 0.002  0.558 49 0.579    1

 4~5
              Diff   Var t-stat DF  pval  holm
Alg_2-Alg_1  0.035 0.010  0.956 49 0.344 0.953
Alg_3-Alg_1  0.030 0.007  1.010 49 0.318 0.953
Alg_3-Alg_2 -0.005 0.004 -0.224 49 0.824 0.953
```

Similarly, ROC curves on a selection of models can be plotted. For e.g., ROC curves resulting from RF analysis on the five binary comparisons:

```
>mcomp.auc = mc.roc(lclas, 1:5, method = "thres")
>plot(mcomp.auc, lcol = c("black", "red", "blue",
+     "green", "purple"), llty = rep(2, 5), leg = "DisId")
```

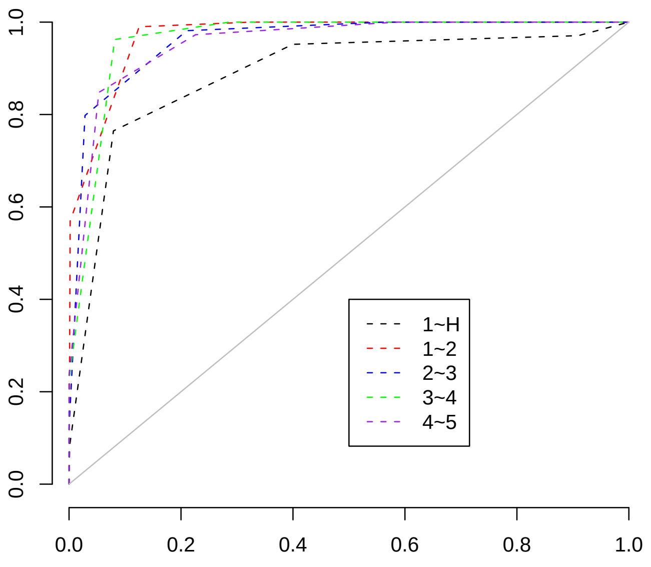

Figure 3: Aggregated ROC curves for all comparisons

Also, this can be done on ROC curves resulting from the three analysis on the first comparison (class 1 vs. class H):

```
>l = which(lclas$cldef[, 7] == "1~H")
>mcomp.auc = mc.roc(lclas, lmod = l, method = "thres")
>plot(mcomp.auc, leg = c("RF", "NLDA", "SVM"))
```

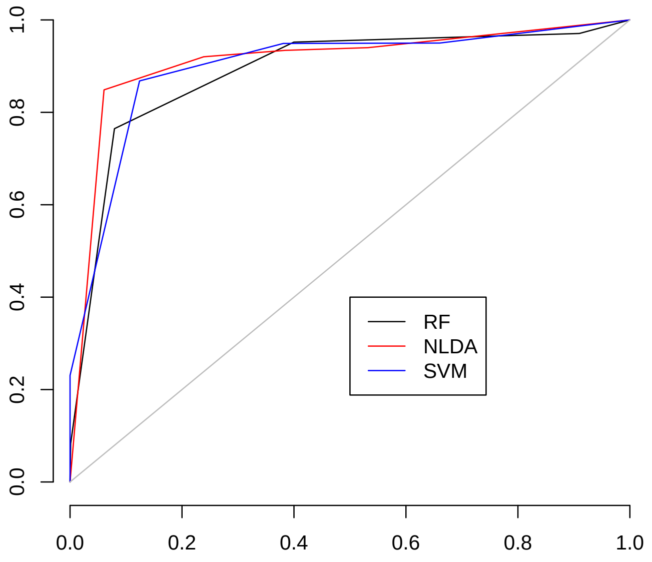

Figure 4: Aggregated ROC curves for the 3 classifiers

6.3 Multiple feature ranking

A similar approach can be undertake to perform multiple resampling based feature rankings on multiple dataset. This time, `feat.rank.re` replaces `accest` and `ft.agg` `mc.agg` to realise and aggregate feature rankings. Further treatment and output of the results are performed with `summ.ftrank` and `tidy.ftrank`.

7 Parallel computation with accest

```
>pack.avai <- installed.packages()
>if (any(pack.avai[, 1] == "snow")) {
+     library(snow)
+     clcomp <- makeCluster(2, type = "MPI")
+     clusterEvalQ(clcomp, library(FIEmspro))
+     clusterExport(clcomp, "accest")
+ }

        2 slaves are spawned successfully. 0 failed.

>if (any(pack.avai[, 1] == "snow")) {
+     ntree = 1000
+     time1 <- system.time(acc1 <- accest(dat, cl,
+         clmeth = "randomForest", pars = pars,
+         tr.idx = tr.idx, ntree = ntree))
+     time2 <- system.time(acc2 <- accest(dat, cl,
+         clmeth = "randomForest", clmpi = NULL,
+         pars = pars, tr.idx = tr.idx, ntree = ntree,
+         seed = 1))
+     time3 <- system.time(acc3 <- accest(dat, cl,
+         clmeth = "randomForest", clmpi = clcomp,
+         pars = pars, tr.idx = tr.idx, ntree = ntree,
+         seed = 1))
+     time4 <- system.time(acc4 <- accest(dat, cl,
+         clmeth = "randomForest", clmpi = clcomp,
+         pars = pars, tr.idx = tr.idx, ntree = ntree,
+         seed = NULL))
+ }

>if (any(pack.avai[, 1] == "snow")) {
+     time = rbind(time1, time2, time3, time4)[,
+        1:3]
+     print(cbind(time, time[, 3]/time[4, 3]))
+ }

      user.self sys.self elapsed
time1    44.205   12.168  61.188 1.3463299
time2    44.285   12.081  63.540 1.3980813
time3     0.773    0.247  44.700 0.9835416
time4     0.767    0.238  45.448 1.0000000
```

```
>if (any(pack.avai[, 1] == "snow")) {
+     lclas = mc.agg(acc2, acc3, acc4)
+     boxplot(mc.meas.iter(lclas, type = "mar"))
+ }
```

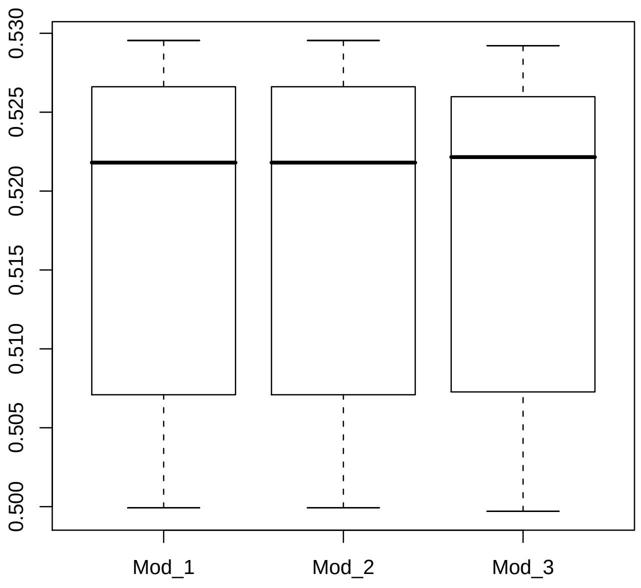

Figure 5: Distributions of the average predictive margins for 3 different accest calls

```
>stopCluster(clcomp)
```

```
[1] 1
```