SML Project

Shubhang Periwal 19201104

4/22/2020

```
data <- read.csv("data_project_deepsolar.csv",header=TRUE)
set.seed(19201104)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(nnet)
library(rpart)
library(kernlab)
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.6.3
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.6.3
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(partykit)
```

```
## Warning: package 'partykit' was built under R version 3.6.3
```

```
## Loading required package: grid
```

```
## Loading required package: libcoin
```

```
## Warning: package 'libcoin' was built under R version 3.6.3
```

```
## Loading required package: mvtnorm
```

```
## Warning: package 'mvtnorm' was built under R version 3.6.2
```

```
library(adabag)
```

```
## Warning: package 'adabag' was built under R version 3.6.3
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.6.2
```

```
## Loading required package: doParallel
```

```
## Warning: package 'doParallel' was built under R version 3.6.3
```

```
## Loading required package: iterators
```

```
## Warning: package 'iterators' was built under R version 3.6.2
```

```
## Loading required package: parallel
```

```
library(lattice)
library(caret)
library(doParallel)
```

```
size = nrow(data)
keep <- sample(1:size,(size*0.7))
test <- setdiff(1:size,keep)#30% for testing
train <- sample(keep,(length(keep)*.80))
val <- setdiff(keep,train)
```

```r
data = data[,c(-79,-76)]#removing derived data
```

```r
corr <- cor(data[train,3:79])
corr[upper.tri(corr)] <- 0
diag(corr)<- 0
temp <- data[1:2]
data <- data[,!apply(corr,2,function(x) any(x > .80))]
dim(data)
```

```
## [1] 20736    58
```

```r
#removing highly related data
```

```r
data <- scale(data)#scaling data
dim(data)
```

```
## [1] 20736    58
```

```r
#computing pca to reduce dimensions, to improve algorithm accuracy
pca <- prcomp(data[train,])
prop <- cumsum(pca$sdev^2)/sum(pca$sdev^2)# compute cumulative proportion of variance
Q <-length( prop[prop<0.95] ) #maintaining atleast 95% of information
Q# only a handful is retained
```

```
## [1] 29
```

```r
#more than 95 % data can be explained using 29 + 2 dimensions, so
#it is unnecessary to  have these many dimensions
```

```r
xz_train <-pca$x[,1:Q]# extract first Q principal components
dat_train <- data.frame(cbind(temp[train,],xz_train))#creating new data frame after reducing dimensions
xz_test <- predict(pca,data[test,])[,1:Q]
dat_test <- data.frame(cbind(temp[test,],xz_test))
xz_val <- predict(pca,data[val,])[,1:Q]
dat_val <- data.frame(cbind(temp[val,],xz_val))
```

```r
data <- data.frame(cbind(temp),predict(pca,data)[,1:Q])
```

```r
fit_glm <- glm(solar_system_count ~., data =  dat_train, family = "binomial")
summary(fit_glm)
```

```
##
## Call:
## glm(formula = solar_system_count ~ ., family = "binomial", data = dat_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6186  -0.3846  -0.1040   0.3739   3.6750
##
```

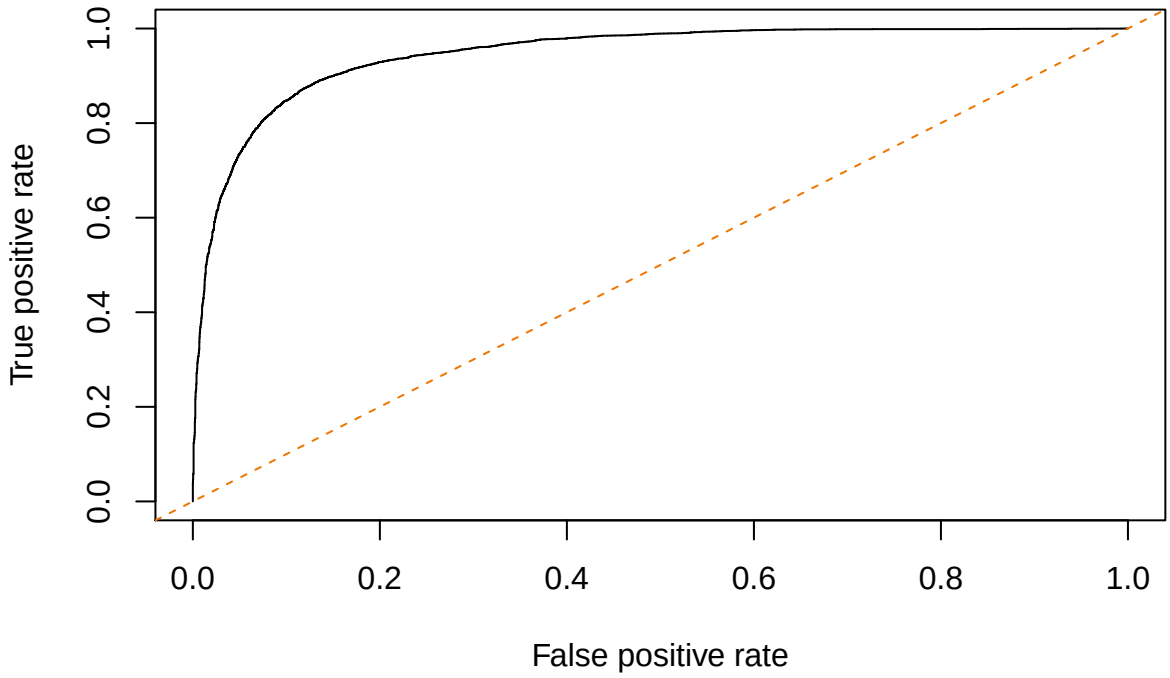
```
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.167791   0.586293  -7.109 1.17e-12 ***
## stateca      4.256197   1.115806   3.814 0.000136 ***
## stateil      4.855023   0.467143  10.393  < 2e-16 ***
## statemi      5.689011   0.491945  11.564  < 2e-16 ***
## statenj      4.574673   0.799342   5.723 1.05e-08 ***
## stateny      5.947717   0.918827   6.473 9.60e-11 ***
## statetx      2.217749   0.323754   6.850 7.38e-12 ***
## PC1         -0.076005   0.032865  -2.313 0.020741 *
## PC2          0.761366   0.160637   4.740 2.14e-06 ***
## PC3         -0.161849   0.056200  -2.880 0.003978 **
## PC4          0.917761   0.145653   6.301 2.96e-10 ***
## PC5          0.396724   0.032891  12.062  < 2e-16 ***
## PC6          0.243640   0.026991   9.027  < 2e-16 ***
## PC7         -0.025672   0.061566  -0.417 0.676696
## PC8          0.747601   0.048109  15.540  < 2e-16 ***
## PC9         -0.021840   0.039472  -0.553 0.580059
## PC10        -0.024945   0.047942  -0.520 0.602846
## PC11         0.072192   0.038092   1.895 0.058066 .
## PC12         0.039518   0.034177   1.156 0.247578
## PC13        -0.302068   0.034324  -8.801  < 2e-16 ***
## PC14         0.189260   0.034522   5.482 4.20e-08 ***
## PC15        -0.036056   0.032475  -1.110 0.266894
## PC16        -0.300119   0.053277  -5.633 1.77e-08 ***
## PC17         0.024841   0.036677   0.677 0.498228
## PC18        -0.159266   0.038446  -4.143 3.43e-05 ***
## PC19         0.131464   0.038649   3.401 0.000670 ***
## PC20         0.005942   0.050051   0.119 0.905505
## PC21         0.310748   0.047067   6.602 4.05e-11 ***
## PC22         0.125477   0.050657   2.477 0.013250 *
## PC23         0.315590   0.045466   6.941 3.88e-12 ***
## PC24        -0.505540   0.043703 -11.568  < 2e-16 ***
## PC25        -0.389519   0.043659  -8.922  < 2e-16 ***
## PC26         0.013247   0.044739   0.296 0.767163
## PC27         0.793435   0.061404  12.922  < 2e-16 ***
## PC28        -0.844479   0.052304 -16.146  < 2e-16 ***
## PC29        -0.412502   0.056142  -7.348 2.02e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 16070.1  on 11611  degrees of freedom
## Residual deviance:  6924.4  on 11576  degrees of freedom
## AIC: 6996.4
##
## Number of Fisher Scoring iterations: 6
```

```r
predf <- predict.glm(fit_glm,newdata = dat_test,type = "response" )
```

```r
y_test_hat <- ifelse(predf > 0.5,"low","high")
table(temp[test,1],y_test_hat)
```

```
##       y_test_hat
##        high  low
##   high 2871  388
##   low   403 2559
```

```
predObj <- prediction(fitted(fit_glm),temp[train,1])
perf <-performance(predObj,"tpr","fpr")
plot(perf)
abline(0,1,col ="darkorange2",lty =2)
```

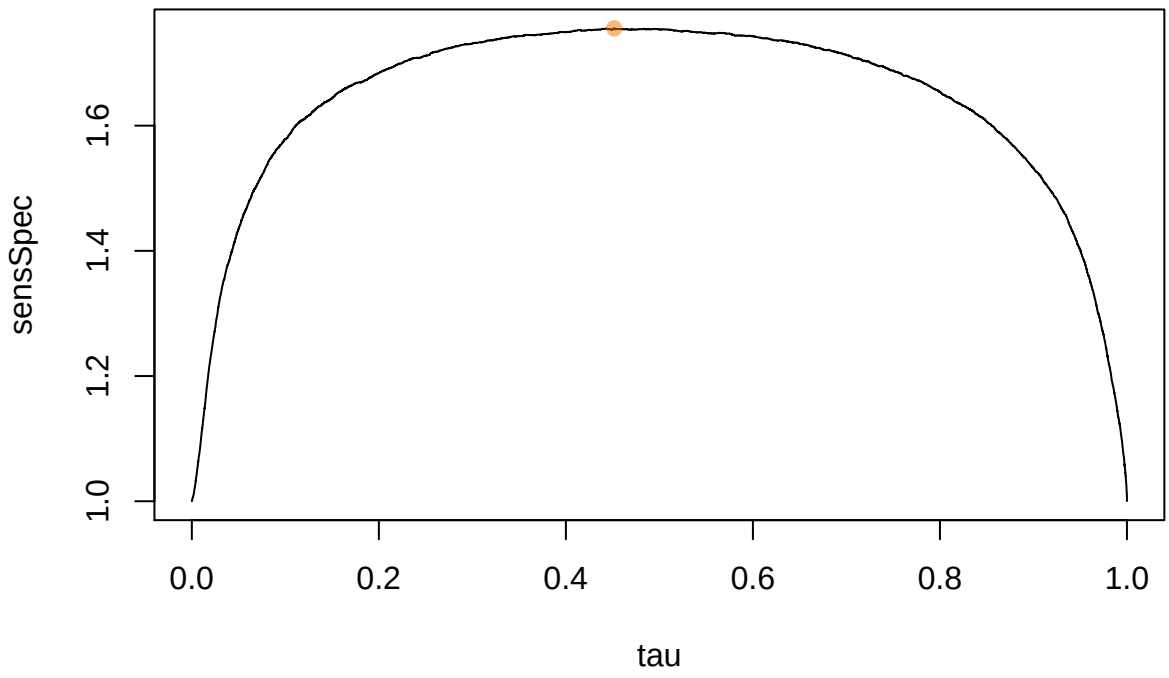



```
#compute area under roc curve
auc <- performance(predObj,"auc")
```

```
auc@y.values
```

```
## [[1]]
## [1] 0.9458324
```

```
sens <-performance(predObj,"sens")
spec <-performance(predObj,"spec")
tau <-sens@x.values[[1]]
sensSpec <-sens@y.values[[1]]+spec@y.values[[1]]
best <-which.max(sensSpec)
plot(tau, sensSpec,type ="l")
points(tau[best], sensSpec[best],pch =19,col=adjustcolor("darkorange2",0.5))
```




```
tau[best]
```

```
##    10707
## 0.451685
```

```
y_test_hat <- ifelse(predf>tau[best],1,0)
table(temp[test,1],y_test_hat)
```

```
##       y_test_hat
##          0    1
##   high 2829  430
##   low   352 2610
```

```
fitRf <- randomForest( solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE) # randomforest
```

```
#  Random Forest
predValRf <- predict(fitRf, type = "class", newdata = dat_test)
tabValRf <- table(dat_test$solar_system_count, predValRf)
tabValRf
```

```
##       predValRf
##        high  low
##   high 2945  314
##   low   354 2608
```

```
accRf <- sum(diag(tabValRf))/sum(tabValRf)
accRf
```

```
## [1] 0.8926218
```

```
fitLog <- multinom(solar_system_count ~ ., data = dat_train,maxit=300, trace=FALSE)
```

```
# Multinomial Regression
predValLog <- predict(fitLog, type = "class", newdata = dat_test)
tabValLog <- table(dat_test$solar_system_count, predValLog)
tabValLog
```

```
##       predValLog
##        high  low
##   high 2871  388
##   low   403 2559
```

```
accLog <- sum(diag(tabValLog))/sum(tabValLog)
accLog
```

```
## [1] 0.87285
```

```
# classification tree

fitRp <- rpart(solar_system_count ~ ., data = dat_train)
```

```
# classification tree
predValRp <- predict(fitRp, type = "class", newdata = dat_test)
tabValRp <- table(dat_test$solar_system_count, predValRp)
tabValRp
```

```
##       predValRp
##        high  low
##   high 2524  735
##   low   342 2620
```

```
accRp <- sum(diag(tabValRp))/sum(tabValRp)
accRp
```

```
## [1] 0.8268767
```

```
#bagging
fitBg <- bagging(solar_system_count ~ ., data = dat_train)
```

```
# bagging
predValBg <- predict(fitBg, type = "class", newdata = dat_test)
tabValBg <- table(dat_test$solar_system_count, predValBg$class)
tabValBg
```

```
##
##        high  low
##   high 2603  656
##   low   363 2599
```

```
accBg <- sum(diag(tabValBg))/sum(tabValBg)
accBg
```

```
## [1] 0.8362
```

```
#boosting
fitBst <- boosting(solar_system_count ~ ., data = dat_train, coeflearn = "Breiman", boos = FALSE)
```

```
# boosting
predValBst <- predict(fitBst, type = "class", newdata = dat_test)
tabValBst <- table(dat_test$solar_system_count, predValBst$class)
tabValBst
```

```
##
##        high  low
##   high 2938  321
##   low   399 2563
```

```
accBst <- sum(diag(tabValBst))/sum(tabValBst)
accBst
```

```
## [1] 0.884263
```

```
#svm
fitSvm <-ksvm(solar_system_count ~ ., data = dat_train)
```

```
#svm
predValSvm <-predict(fitSvm,newdata =dat_test)
tabValSvm <-table(dat_test$solar_system_count, predValSvm)
tabValSvm
```

```
##       predValSvm
##        high  low
##   high 2926  333
##   low   338 2624
```

```
accSvm <-sum(diag(tabValSvm))/sum(tabValSvm)
accSvm
```

```
## [1] 0.8921395
```

```
fin_res <- cbind(predValRf,predValSvm, predValBg$class)
for(i in 1:6221)
{
  if(fin_res[i,1]=="2")
```

```
    fin_res[i,1] = "low"
  if(fin_res[i,2]=="2")
    fin_res[i,2] = "low"
  if(fin_res[i,1]=="1")
    fin_res[i,1] = "high"
  if(fin_res[i,2]=="1")
    fin_res[i,2] = "high"
}

res <- apply(fin_res,1,function(x){names(which.max(table(x)))})

tabVal <- table(data[test,]$solar_system_count,res)
tabVal
```

```
##       res
##        high  low
##   high 2911  348
##   low   352 2610
```

```
accMax <- sum(diag(tabVal))/sum(tabVal)
accMax
```

```
## [1] 0.8874779
```

```
# replicate the process a number of times
R <- 50
out <- matrix(NA, R, 4)
colnames(out) <- c("val_random_forest", "val_logistic", "best", "test")
out <- as.data.frame(out)

for ( r in 1:R ) {

  size = nrow(data)
  keep <- sample(1:size,(size*0.7))
  test <- setdiff(1:size,keep)#30% for testing
  train <- sample(keep,(length(keep)*.80))
  val <- setdiff(keep,train)

  # fit classifiers to only the training data

  fitRf <- randomForest( solar_system_count ~ ., data = data[train,], trace=FALSE) #random forest
  fitLog <- multinom(solar_system_count ~ ., data = data[train,], trace=FALSE) #multinomial
  fitBg <- bagging(solar_system_count ~ ., data = data[train,])#bagging
  fitSvm <-ksvm(solar_system_count ~ ., data = data[train,]) #SVM
  fitBst <- boosting(solar_system_count ~ ., data = data[train,], coeflearn = "Breiman", boos = FALSE)#
  fitRp <- rpart(solar_system_count ~ ., data = data[train,]) #classification tree

  # classify the validation data observations
  #  Random Forest
  predValRf <- predict(fitRf, type = "class", newdata = data[val,])
  tabValRf <- table(data[val,]$solar_system_count, predValRf)
  #tabValRf
```

```
  accRf <- sum(diag(tabValRf))/sum(tabValRf)

  # Multinomial Regression
predValLog <- predict(fitLog, type = "class", newdata = data[val,])
tabValLog <- table(data[val,]$solar_system_count, predValLog)
#tabValLog
accLog <- sum(diag(tabValLog))/sum(tabValLog)
accLog

# classification tree
predValRp <- predict(fitRp, type = "class", newdata = data[val,])
tabValRp <- table(data[val,]$solar_system_count, predValRp)
#tabValRp
accRp <- sum(diag(tabValRp))/sum(tabValRp)
accRp

# boosting
predValBst <- predict(fitBst, type = "class", newdata = data[val,])
tabValBst <- table(data[val,]$solar_system_count, predValBst$class)
#tabValBst
accBst <- sum(diag(tabValBst))/sum(tabValBst)
accBst

# bagging
predValBg <- predict(fitBg, type = "class", newdata = data[val,])
tabValBg <- table(data[val,]$solar_system_count, predValBg$class)
#tabValBg
accBg <- sum(diag(tabValBg))/sum(tabValBg)
accBg


  #svm
predValSvm <-predict(fitSvm,newdata =data[val,])
tabValSvm <-table(data[val,]$solar_system_count, predValSvm)
#tabValSvm
accSvm <-sum(diag(tabValSvm))/sum(tabValSvm)
accSvm


  # accuracy
  acc <- c(random_Forest = accRf, multinomial = accLog, classificationTree = accRp, Boosting = accBst, B
  out[r,1] <- accRf
  out[r,2] <- accLog
  out[r,3] <- accRp
  out[r,4] <- accBst
  out[r,5] <- accBg
  out[r,6] <- accSvm


  # use the method that did best on the validation data
  # to predict the test data
  best <- names( which.max(acc) )
  switch(best,
```

```
        multinomial = {
          predTestLog <- predict(fitLog, type = "class", newdata = data[test,])
          tabTestLog <- table(data[test,]$classes, predTestLog)
          accBest <- sum(diag(tabTestLog))/sum(tabTestLog)
        },
        #  Random Forest
  random_Forest = {
  predValRf <- predict(fitRf, type = "class", newdata = data[test,])
  tabValRf <- table(data[test,]$solar_system_count, predValRf)
  #tabValRf
  accBest <- sum(diag(tabValRf))/sum(tabValRf)
  },
  # Multinomial Regression
  multinomial = {
    predValLog <- predict(fitLog, type = "class", newdata = data[test,])
tabValLog <- table(data[test,]$solar_system_count, predValLog)
#tabValLog
accBest <- sum(diag(tabValLog))/sum(tabValLog)
  },

# classification tree
classificationTree = {
predValRp <- predict(fitRp, type = "class", newdata = data[test,])
tabValRp <- table(data[test,]$solar_system_count, predValRp)
#tabValRp
accBest <- sum(diag(tabValRp))/sum(tabValRp)
},

# boosting
Boosting = {
predValBst <- predict(fitBst, type = "class", newdata = data[test,])
tabValBst <- table(dat_test$solar_system_count, predValBst$class)
#tabValBst
accBest <- sum(diag(tabValBst))/sum(tabValBst)
},


# bagging
Bagging = {
predValBg <- predict(fitBg, type = "class", newdata = data[test,])
tabValBg <- table(data[test,]$solar_system_count, predValBg$class)
#tabValBg
accBest <- sum(diag(tabValBg))/sum(tabValBg)
},

SVM = {
  #svm
predValSvm <-predict(fitSvm,newdata =data[test,])
tabValSvm <-table(data[test,]$solar_system_count, predValSvm)
#tabValSvm
accBest <-sum(diag(tabValSvm))/sum(tabValSvm)

}
```

```
  )
  out[r,7] <- best
  out[r,8] <- accBest

}
```

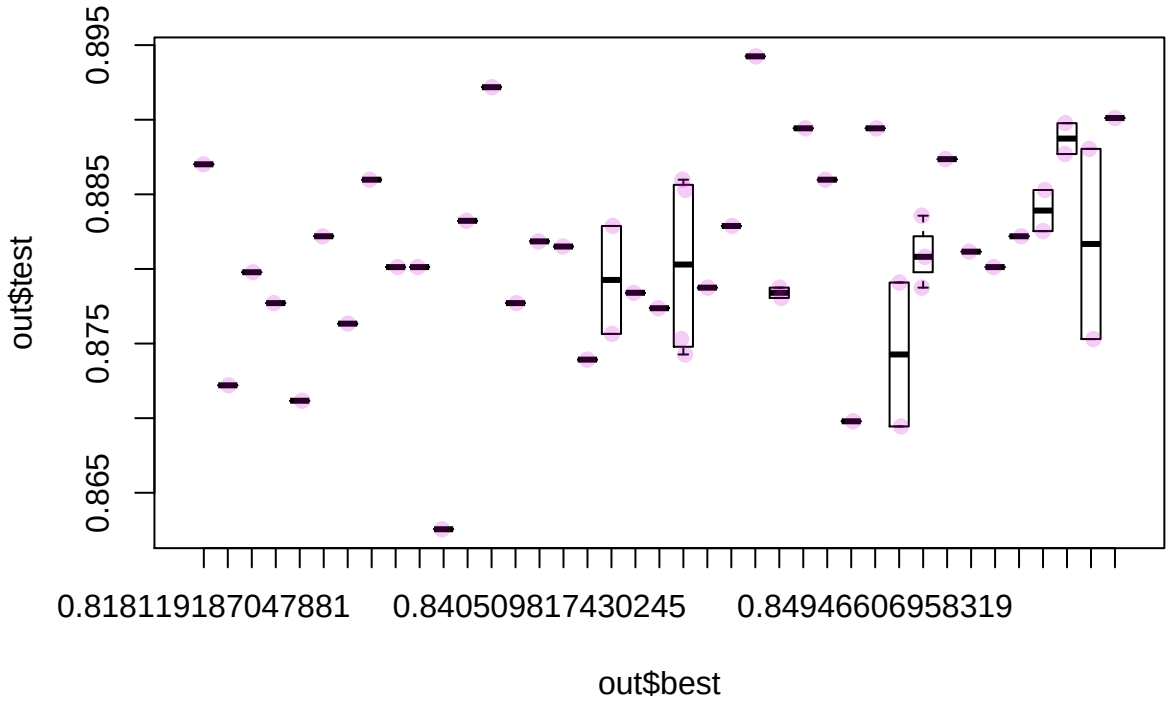
```
# check out the error rate summary statistics
table(out[,7])
```

```
##
##      Boosting random_Forest           SVM
##             1             7            42
```

```
tapply(out[,8], out[,7], summary)
```

```
## $Boosting
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.7078  0.7078  0.7078  0.7078  0.7078  0.7078
##
## $random_Forest
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8876  0.8900  0.8915  0.8915  0.8929  0.8954
##
## $SVM
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8844  0.8905  0.8938  0.8931  0.8957  0.9029
```

```
boxplot(out$test ~ out$best)
stripchart(out$test ~ out$best, add = TRUE, vertical = TRUE,
           method = "jitter", pch = 19, col = adjustcolor("magenta3", 0.2))
```




```
#avg <- t( sapply(out[1:6], colMeans))
avg <- out[,1:6]
meanAcc <- colMeans(out[,1:6]) # estimated mean accuracy
meanAcc
```

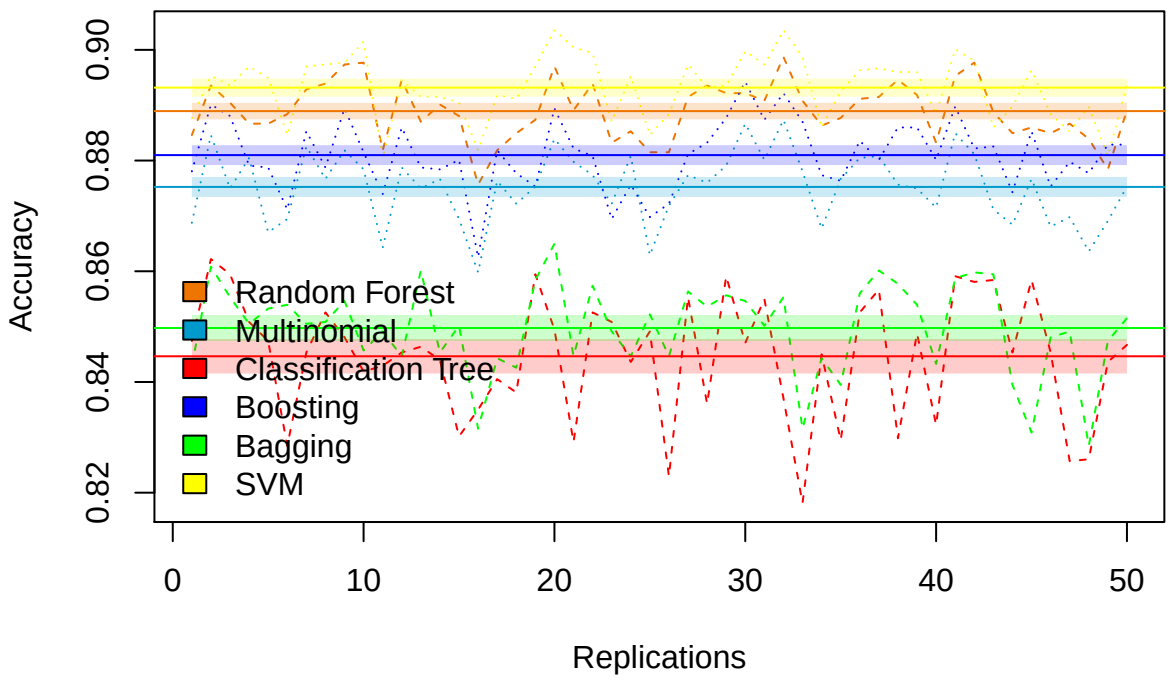
```
## val_random_forest      val_logistic              best              test
##         0.8889356         0.8752394         0.8446435         0.8809852
##                V5                V6
##         0.8497485         0.8931795
```

```
sdAcc <- apply(avg, 2, sd)/sqrt(R) # estimated mean accuracy standard deviation
sdAcc
```

```
## val_random_forest      val_logistic              best              test
##      0.0007460555      0.0009029868      0.0015586503      0.0008959631
##                V5                V6
##      0.0011744105      0.0008168445
```

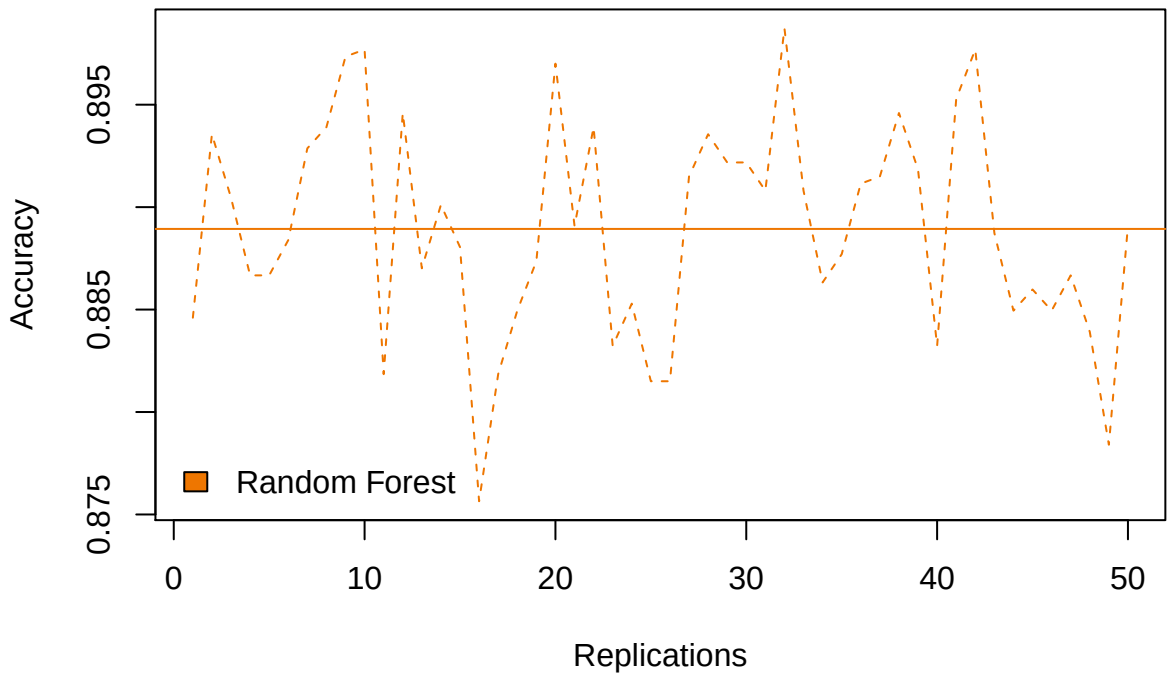
```
matplot(avg, type = "l", lty = c(2,3), col = c("darkorange2", "deepskyblue3","red","blue","green","yell
xlab = "Replications", ylab = "Accuracy")
#
# add confidence intervals
bounds1 <- rep( c(meanAcc[1] - 2*sdAcc[1], meanAcc[1] + 2*sdAcc[1]), each = R )
bounds2 <- rep( c(meanAcc[2] - 2*sdAcc[2], meanAcc[2] + 2*sdAcc[2]), each = R )
```

```
bounds3 <- rep( c(meanAcc[3] - 2*sdAcc[3], meanAcc[3] + 2*sdAcc[3]), each = R )
bounds4 <- rep( c(meanAcc[4] - 2*sdAcc[4], meanAcc[4] + 2*sdAcc[4]), each = R )
bounds5 <- rep( c(meanAcc[5] - 2*sdAcc[5], meanAcc[5] + 2*sdAcc[5]), each = R )
bounds6 <- rep( c(meanAcc[6] - 2*sdAcc[6], meanAcc[6] + 2*sdAcc[6]), each = R )
polygon(c(1:R, R:1), bounds1, col = adjustcolor("darkorange2", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds2, col = adjustcolor("deepskyblue3", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds3, col = adjustcolor("red", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds4, col = adjustcolor("blue", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds5, col = adjustcolor("green", 0.2), border = FALSE)
polygon(c(1:R, R:1), bounds6, col = adjustcolor("yellow", 0.2), border = FALSE)
#
# add estimated mean line
abline(h = meanAcc, col = c("darkorange2", "deepskyblue3","red","blue","green","yellow"))
#
# add legend
legend("bottomleft", fill = c("darkorange2", "deepskyblue3","red","blue","green","yellow"),
legend = c("Random Forest", "Multinomial","Classification Tree","Boosting","Bagging","SVM"), bty = "n")
```



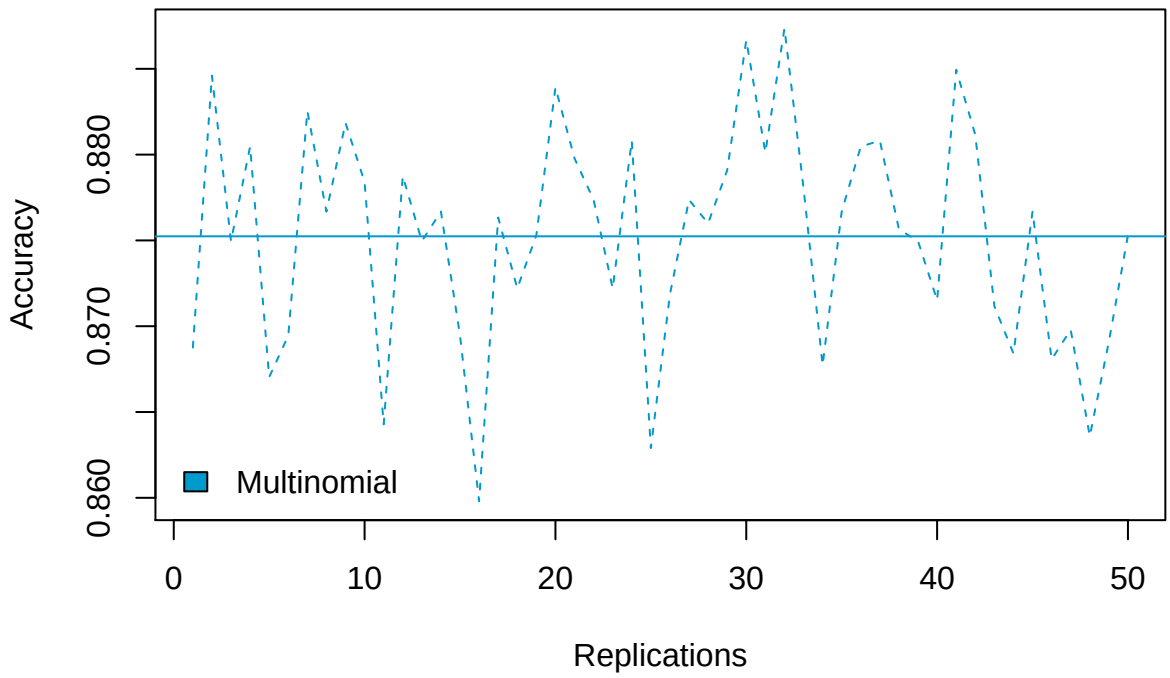

```
#Random Forest
matplot(avg[1], type = "l", lty = c(2,3), col = c("darkorange2"),
xlab = "Replications", ylab = "Accuracy")
#
# add estimated mean line
abline(h = meanAcc[1], col = c("darkorange2"))
#
```

```
# add legend
legend("bottomleft", fill = c("darkorange2"),
legend = c("Random Forest"), bty = "n")
```



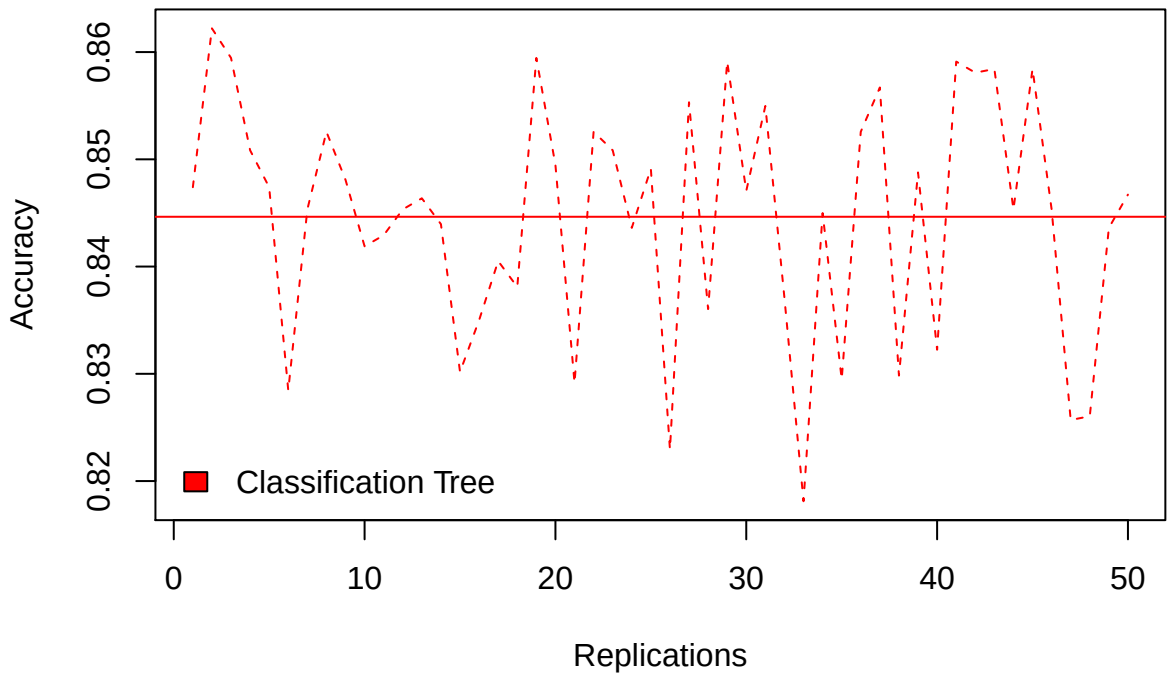

```
#Multinomial
matplot(avg[2], type = "l", lty = c(2,3), col = c("deepskyblue3"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[2], col = c("deepskyblue3"))
#
# add legend
legend("bottomleft", fill = c("deepskyblue3"),
legend = c("Multinomial"), bty = "n")
```

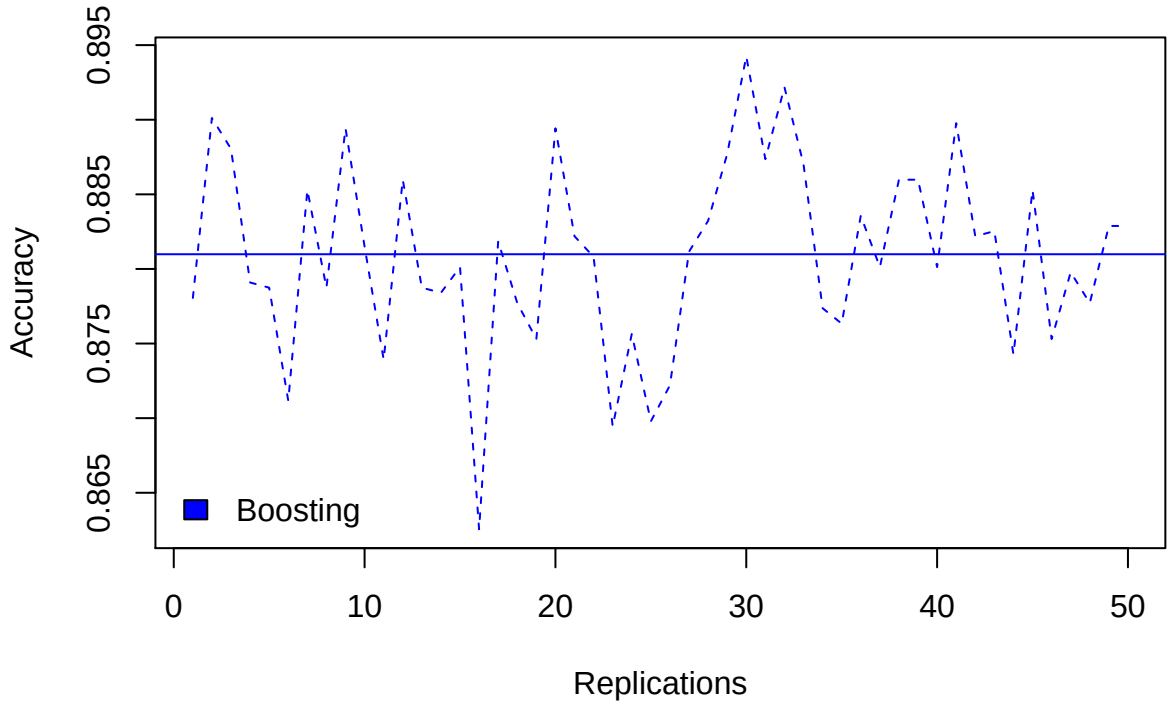



```
#Classification tree
matplot(avg[3], type = "l", lty = c(2,3), col = c("red"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[3], col = c("red"))
#
# add legend
legend("bottomleft", fill = c("red"),
legend = c("Classification Tree"), bty = "n")
```

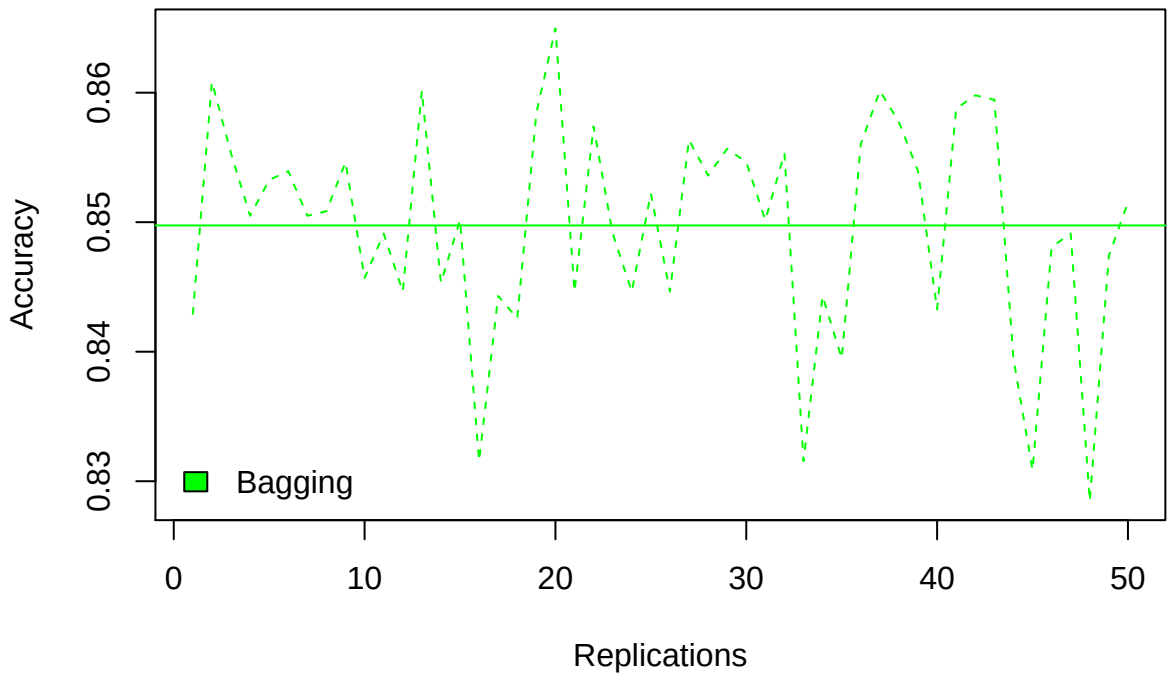



```
#Boosting
matplot(avg[4], type = "l", lty = c(2,3), col = c("blue"),
xlab = "Replications", ylab = "Accuracy")
#
# add estimated mean line
abline(h = meanAcc[4], col = c("blue"))
#
# add legend
legend("bottomleft", fill = c("blue"),
legend = c("Boosting"), bty = "n")
```



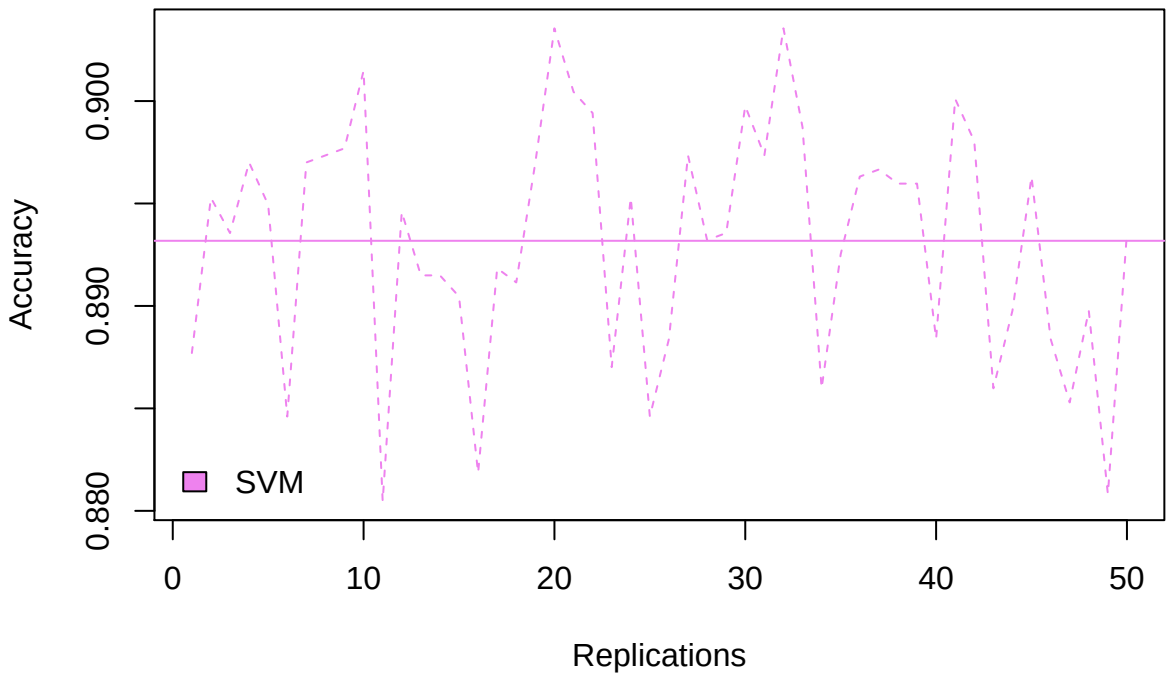

```
#Bagging
matplot(avg[5], type = "l", lty = c(2,3), col = c("green"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[5], col = c("green"))
#
# add legend
legend("bottomleft", fill = c("green"),
legend = c("Bagging"), bty = "n")
```




```
#SVM
matplot(avg[6], type = "l", lty = c(2,3), col = c("violet"),
xlab = "Replications", ylab = "Accuracy")

# add estimated mean line
abline(h = meanAcc[6], col = c("violet"))
#
# add legend
legend("bottomleft", fill = c("violet"),
legend = c("SVM"), bty = "n")
```


Accuracy
0.880
0.890
0.900
SVM
0
10
20
30
40
50
Replications