Comparing speed of packages for computing ROC curves

Toby Dylan Hocking

July 12, 2017

1 Introduction

The goal of this vignette is to compare how long it takes to compute the ROC curves and AUC using different R packages: WeightedROC, ROCR, pROC, glmnet.

2 Data

The data set I will analyze is the `spam` data set discussed in the book "The Elements of Statistical Learning" by Hastie, Tibshirani, and Friedman. The book is available to read for free as a downloadable PDF at

`http://statweb.stanford.edu/~tibs/ElemStatLearn/`

The data set is available in the R package ElemStatLearn:

```
> library(ElemStatLearn)
> data(spam)
> is.label <- names(spam) == "spam"
> X <- as.matrix(spam[,!is.label])
> y <- spam[,is.label]
> set.seed(1)
> train.i <- sample(nrow(spam), nrow(spam)/2)
> sets <-
+   list(train=list(features=X[train.i, ], label=y[train.i]),
+        test=list(features=X[-train.i, ], label=y[-train.i]))
> str(sets)
```

```
List of 2
 $ train:List of 2
  ..$ features: num [1:2300, 1:57] 0 0 0 0 0 0 0 0 0 0.4 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2300] "1222" "1712" "2635" "4176" ...
  .. .. ..$ : chr [1:57] "A.1" "A.2" "A.3" "A.4" ...
  ..$ label   : Factor w/ 2 levels "email","spam": 2 2 1 1 2 1 1 1 1 2 ...
 $ test :List of 2
  ..$ features: num [1:2301, 1:57] 0 0 0.06 0 0 0 0 0 0 0.05 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2301] "1" "7" "10" "12" ...
  .. .. ..$ : chr [1:57] "A.1" "A.2" "A.3" "A.4" ...
  ..$ label   : Factor w/ 2 levels "email","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

I divided the data set into half train, half test. I will fit a model on the training set and see if it works on the test set.

3 Model fitting

Below, I fit an L1-regularized logistic regression model to the spam training set.

```
> library(glmnet)
> system.time({
+   fit <- cv.glmnet(sets$train$features, sets$train$label, family="binomial")
+ })

   user  system elapsed 
 10.536   0.052  10.643 
```

On my Intel i7 2.8GHz CPU, it took about 10 seconds to fit the model.

4 Timing test ROC curve computation

ROC analysis is useful for evaluating binary classifiers. Below we compute the ROC curves using WeightedROC, pROC, and ROCR.

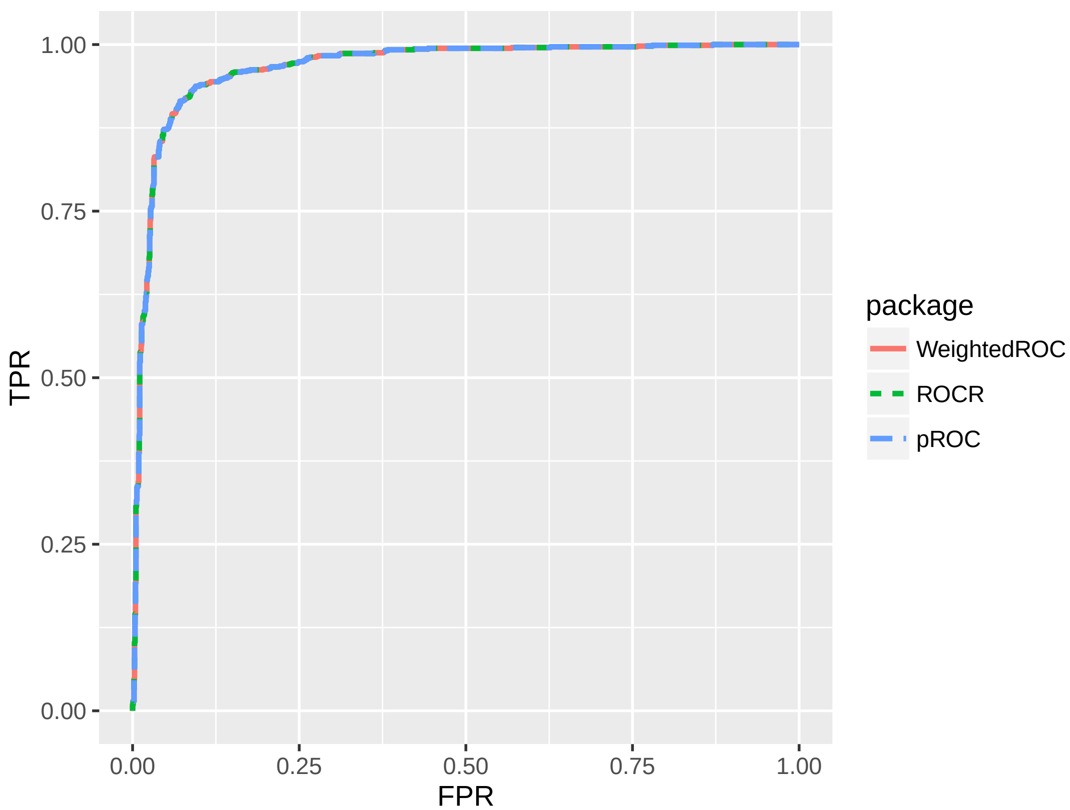
```
> library(WeightedROC)
> library(ROCR)
> library(pROC)
> set <- sets$test
> guess <- predict(fit, set$features)
> if(require(microbenchmark)){
+   microbenchmark(WeightedROC={
+     wroc <- WeightedROC(guess, set$label)
+   }, ROCR={
+     pred <- prediction(guess, set$label)
+     perf <- performance(pred, "tpr", "fpr")
+   }, pROC={
+     proc <- roc(set$label, guess, algorithm=2)
+   })
+ }else{
+   wroc <- WeightedROC(guess, set$label)
+   pred <- prediction(guess, set$label)
+   perf <- performance(pred, "tpr", "fpr")
+   proc <- roc(set$label, guess, algorithm=2)
+ }

Unit: milliseconds
        expr       min        lq      mean    median        uq       max neval
 WeightedROC  2.788754  3.046869  3.193505  3.070302  3.097944  7.385582   100
        ROCR  5.717512  6.136067  6.595780  6.198159  6.364028  9.283899   100
        pROC 26.544743 27.054944 27.760863 27.329892 28.033447 31.703999   100
 cld
 a  
  b 
   c
```

It is clear that WeightedROC is the fastest computation (on my computer, median 1.8 milliseconds), followed by ROCR (7.2 milliseconds), and finally pROC (26.4 milliseconds). However, all of the ROC computations are much faster than the model fitting (10 seconds). Below, we plot the ROC curves.

```
> perfDF <- function(p){
+   data.frame(FPR=p@x.values[[1]], TPR=p@y.values[[1]], package="ROCR")
+ }
> procDF <- function(p){
+   data.frame(FPR=1-p$specificities, TPR=p$sensitivities, package="pROC")
+ }
> roc.curves <-
```

```
+   rbind(data.frame(wroc[,c("FPR", "TPR")], package="WeightedROC"),
+         perfDF(perf), procDF(proc))
> library(ggplot2)
> rocPlot <- ggplot()+
+   geom_path(aes(FPR, TPR, color=package, linetype=package),
+             data=roc.curves, size=1)+
+   coord_equal()
> print(rocPlot)
>
```

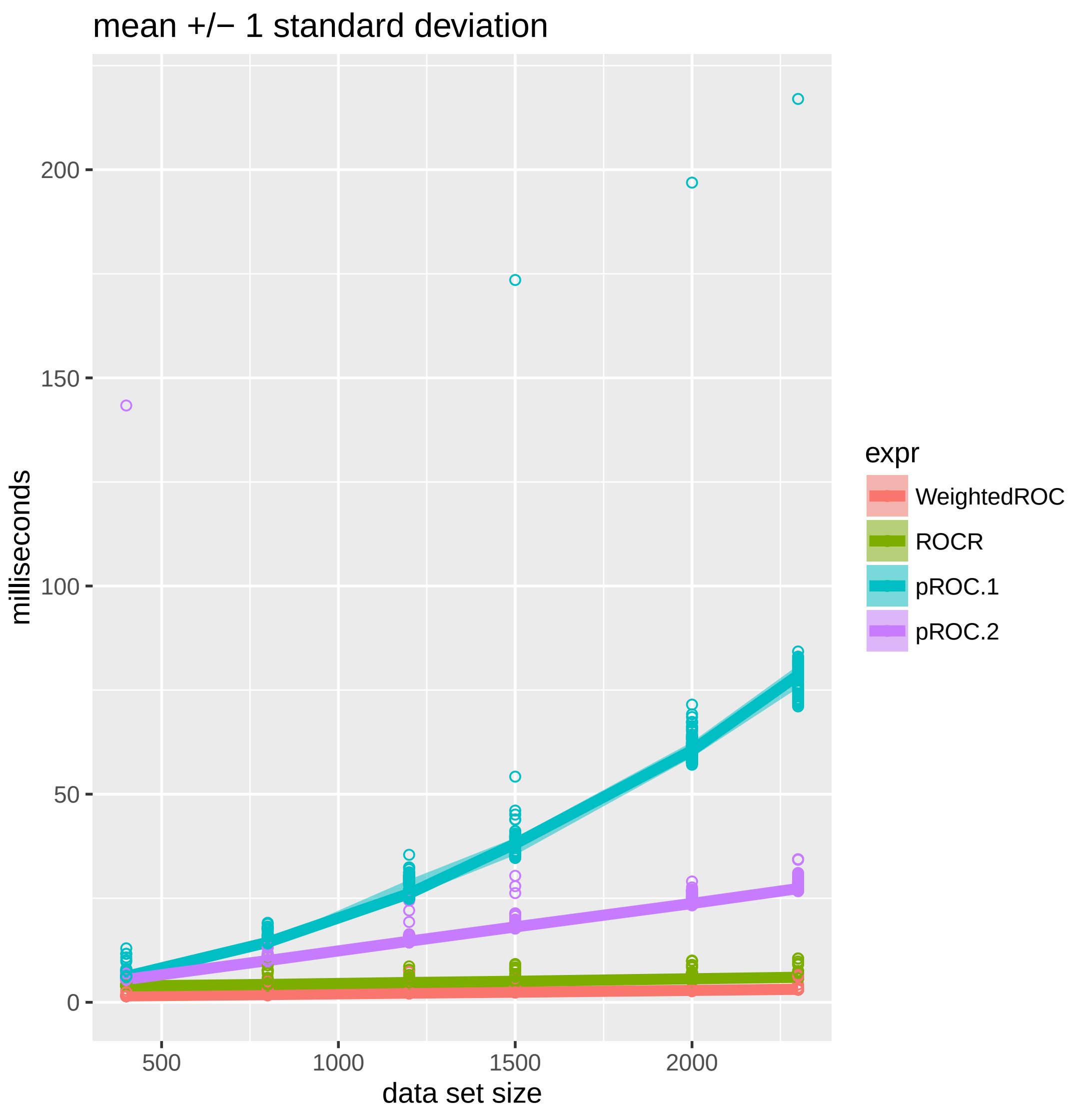


5 Scaling

In this section I was interested in seeing if there are any differences between algorithms as the number of data points changes.

```
> if(require(microbenchmark)){
+   stats.by.size.expr <- list()
+   ms.by.size <- list()
+   for(size in c(400, 800, 1200, 1500, 2000, 2300)){
+     indices <- seq(1, length(set$label), l=size)
```

```
+     y <- set$label[indices]
+     y.hat <- guess[indices]
+     this.size <- microbenchmark(WeightedROC={
+       wroc <- WeightedROC(y.hat, y)
+     }, ROCR={
+       pred <- prediction(y.hat, y)
+       perf <- performance(pred, "tpr", "fpr")
+     }, pROC.1={
+       proc <- roc(y, y.hat, algorithm=1)
+     }, pROC.2={
+       proc <- roc(y, y.hat, algorithm=2)
+     })
+     this.size$milliseconds <- this.size$time/1e6
+     ms.by.size[[paste(size)]] <- data.frame(size, this.size)
+     this.by.expr <- split(this.size, this.size$expr)
+     for(expr in names(this.by.expr)){
+       stats <- with(this.by.expr[[expr]], {
+         data.frame(median=median(milliseconds),
+                    q25=quantile(milliseconds, 0.25),
+                    q75=quantile(milliseconds, 0.75))
+       })
+       stats.by.size.expr[[paste(size, expr)]] <- data.frame(size, expr, stats)
+     }
+   }
+   ms <- do.call(rbind, ms.by.size)
+   algo.stats <- do.call(rbind, stats.by.size.expr)
+   timePlot <- ggplot()+
+     geom_ribbon(aes(size, ymin=q25, ymax=q75, fill=expr),
+                 data=algo.stats, alpha=1/2)+
+     geom_line(aes(size, median, color=expr), data=algo.stats, size=2)+
+     geom_point(aes(size, milliseconds, color=expr), data=ms, pch=1)+
+     ylab("milliseconds")+
+     xlab("data set size")+
+     ggtitle("mean +/- 1 standard deviation")
+   print(timePlot)
+ }
```



The figure above shows that for the spam data, the data set size does not affect the speed ordering of the algorithms for ROC curve computation. In all cases, WeightedROC is fastest, followed by ROCR, then pROC.2, then pROC.1. It makes sense that pROC.2 is faster than pROC.1, since pROC.2 uses the cumsum function, but pROC.1 does not.