Simple_h2o

Rob McCulloch

April 15, 2019

Contents

Start Up h2o

To use h2o you have to imagine that all the work is being done on a remote server you are communicating with. For example, we have to load the library and then initialize the server.

First we load the library as usual:

```
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------

##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

Then we initialize the server:

```
h2oServer = h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmpiNtzP2/h2o_root_started_from_r.out
##     /tmp/RtmpiNtzP2/h2o_root_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 seconds 207 milliseconds
##     H2O cluster timezone:       America/Phoenix
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.20.0.8
##     H2O cluster version age:    6 months and 25 days !!!
##     H2O cluster name:           H2O_started_from_R_root_zyo008
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   6.84 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.1 (2018-07-02)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (6 months and 25 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

Notice that on Rob's machine 8 cores are detected and used.
h2o will use all the cores!!

Setup up the data

Now let's make some toy data with a binary y.
I'll use that boston housing data again but let y =1 if price is above the median and 0 else.

```
library(MASS)
attach(Boston)
y = Boston$medv

## let's make y binary
```

```r
y = as.factor(y>median(y))

## for x we will use dis and lstat
x  = cbind(Boston$dis,Boston$lstat)
p = ncol(x)
for(i in 1:p) {
   rgx = range(x)
   x[,i] = (x[,i]-rgx[1])/(rgx[2]-rgx[1])
}
colnames(x) = c("dis","lstat")

## data as a data frame
dfd = data.frame(y,x)

## train and test
set.seed(99)
n=nrow(dfd)
ii = sample(1:n,floor(.75*n))
dftr = dfd[ii,]; ytr = y[ii] #train
dfte = dfd[-ii,] ; yte = y[-ii] #test
```

Run a logit for comparison

Let's first run a simple logit and see what we get.

```r
glmf = glm(y~.,data=dftr,family=binomial)
yhltr = predict(glmf,type="response")
yhlte = predict(glmf,dfte,type="response")

##in-sample confusion matrix
table(dftr$y,yhltr>.5)
```

```
##
##          FALSE TRUE
##   FALSE   142   38
##   TRUE     26  173
```

```r
##out-of-sample confusion matrix
ocfl = table(dfte$y,yhlte>.5)
ocfl
```

```
##
##          FALSE TRUE
##   FALSE    57   19
##   TRUE      8   43
```

```r
cat("% wrong out of sample is",1-sum(diag(ocfl))/sum(ocfl))
```

```
## % wrong out of sample is 0.2125984
```

Put the data in h2o form

We can look and see what is currently on the server:

```
h2o.ls()
```

```
## [1] key
## <0 rows> (or 0-length row.names)
```

Right now there is nothing.

Let's put our data on the server (or cluster) so we can fit our models using h2o.

```
dftrain = as.h2o(dftr, destination_frame = "bost.train")
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
dftest = as.h2o(dfte, destination_frame = "bost.test")
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%

# see that bost.test and bost.train are now on server
h2o.ls()
##          key
## 1  bost.test
## 2 bost.train
```

Now bost.train and bost.test show up on the server.

dftrain is the R object we use to access bost.train.

dftest is the R object we use to access bost.test.

As usual we can print dftrain just by typing its name.

```
dftrain #h2o prints out first 6 rows
```

```
##       y        dis     lstat
## 1  TRUE 0.13112778 0.1651304
## 2  TRUE 0.19530733 0.1040295
## 3  TRUE 0.14483556 0.1214116
## 4 FALSE 0.03448117 0.3976824
## 5  TRUE 0.04729590 0.0832236
```

```
## 6 FALSE 0.07042269 0.3513300
##
## [379 rows x 3 columns]
```

There are two kinds of *classes* in R, S3 and S4.

S3 is a very simple setup.
dftrain and dftest are S3 classes.

```r
cat("is dh2o an S4 class?:\n")
```

```
## is dh2o an S4 class?:
```

```r
isS4(dftrain)
```

```
## [1] FALSE
```

```r
cat("it is an S3 class with class name:\n")
```

```
## it is an S3 class with class name:
```

```r
print(class(dftrain))
```

```
## [1] "H2OFrame"
```

We see that dftrain is not S4 but it is S3 with class name H2OFrame.

A simple way to see what information is in the S3 class is to use the *attributes*.

```r
temp = attributes(dftrain)
is.list(temp)
```

```
## [1] TRUE
```

```r
names(temp)
```

```
## [1] "class" "op"    "id"    "eval"  "nrow"  "ncol"  "types" "data"
```

```r
cat("the h2o id of dftrain is: ",attr(dftrain,"id"),"\n")
```

```
## the h2o id of dftrain is:  bost.train
```

```r
cat("the class name  of dftrain is: ",attr(dftrain,"class"),"\n")
```

```
## the class name  of dftrain is:  H2OFrame
```

```r
cat("the number of rows of dftrain is: ",attr(dftrain,"nrow"),"\n")
```

```
## the number of rows of dftrain is:  379
```

If I just print out temp or do str(dftrain) I will get a ton of information.

Fit a Deep Neural Net

Ok, let's try a deep neural net.

Remember, "deep" just means we have more than one hidden layer.

I'll do *hidden=c(20,10)* which means two hidden layers where the first layer has 20 units and the second layer has 10 units.

```
########################################################################
#  2 hidden layer 10 neurons

nnf = h2o.deeplearning(x=2:3, y=1,
                         training_frame = dftrain,
                         hidden = c(20,10),
                         activation = "Tanh",
                         epochs = 200,
                         model_id = "boston.nn_20-10"
                         )
```

```
##
  |
  |                                                                  |   0%
  |
  |==================================================================| 100%
```

```
#nnf is an S4 class:
cat("is model object nnf S4?:\n",isS4(nnf))
```

```
## is model object nnf S4?:
##  TRUE
```

```
#It is S4, to pull of a slot use @
cat("h2o model_id of nnf is",nnf@model_id,"\n")
```

```
## h2o model_id of nnf is boston.nn_20-10
```

```
#check this using h2o.ls
print(h2o.ls())
```

```
##                                                                        key
## 1                                         _b43aeaf0d6fe0b6337b8cb2b70304ddd
## 2                                                                bost.test
## 3                                                               bost.train
## 4                                                          boston.nn_20-10
## 5 modelmetrics_boston.nn_20-10@-2834169925089100188_on_bost.train@6903539168338962452
```

```
## to see the whole thing which is a lot:
#print(str(nnf))
```

We can use a predict function. Let's first get the out-of-sample predictions using dftest.

```
phato = h2o.predict(nnf,dftest)
```

```
##
```

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```r
dim(phato)
```

```
## [1] 127   3
```

```r
names(phato)
```

```
## [1] "predict" "FALSE"   "TRUE"
```

The first column is the prediction, the second column is $p(y = false|x)$ and the third column is $p(y = true|x)$.

Let's just pull off the third column and convert it to an R data structure.

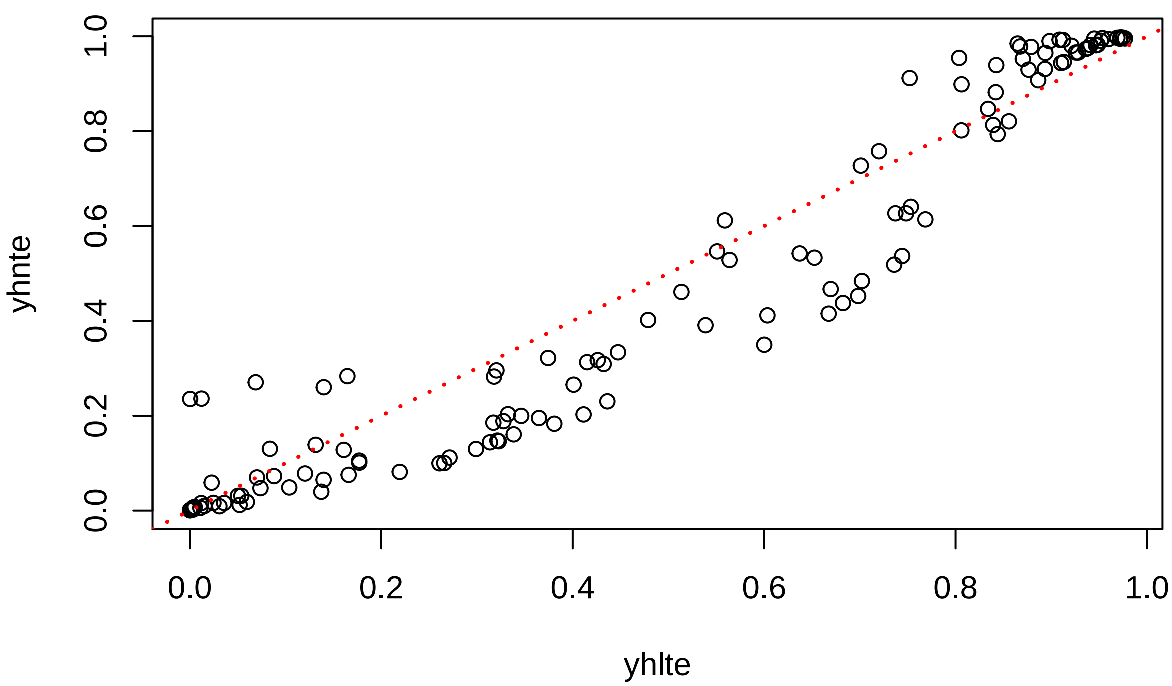
```r
yhnte = as.matrix(phato[,3])[,1]
```

The as.matrix converts it to R, and the [,1] coverts the matrix with one column to a double vector.

Ok now we can compare neural nets to logit!!

The yhnte is comparable to the yhlte we got from the logit fit.

```r
plot(yhlte,yhnte)
abline(0,1,col="red",lwd=2,lty=3)
```



Let's look at the confusion matrices.

```r
ocfn = table(dfte$y,yhnte>.5)
ocfn
```

```
##
##         FALSE TRUE
##   FALSE    65   11
##   TRUE      9   42
```

```
cat("neural net, % wrong out of sample is",1-sum(diag(ocfn))/sum(ocfn))
```

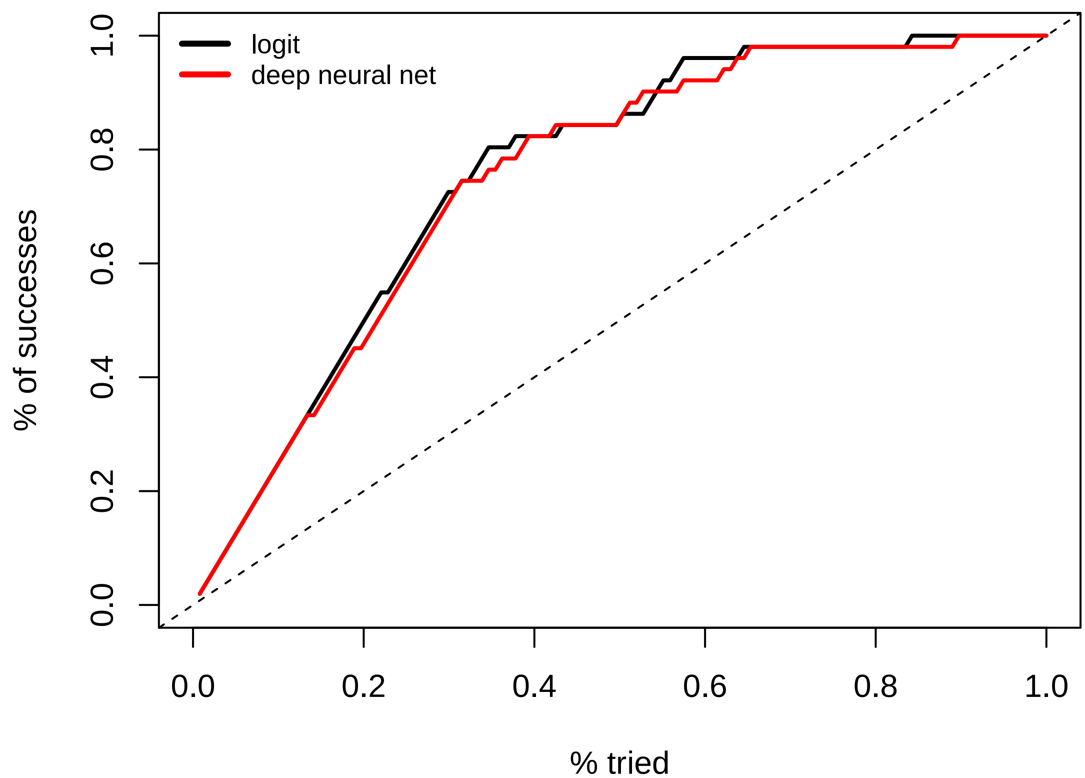
```
## neural net, % wrong out of sample is 0.1574803
```

```
cat("logit, % wrong out of sample is",1-sum(diag(ocfl))/sum(ocfl))
```

```
## logit, % wrong out of sample is 0.2125984
```

Let's look at the lift curves.

```
source("http://www.rob-mcculloch.org/2019_ml/webpage/notes/lift-loss.R")
yhatL = list(yhlte,yhnte)
lift.many.plot(yhatL,dfte$y)
legend("topleft",legend=c("logit","deep neural net"),lwd=rep(3,1),col=1:2,bty="n",cex=.8)
```



Not too different.

This is just a toy example, but a deep neural net does ok!!

Amazing given the complexity of the model.

Note that h2o computes a huge amount of performance statistics for you.

```
print(h2o.confusionMatrix(nnf,thresholds=.5))
```

```
## Warning in h2o.find_row_by_threshold(object, t): Could not find exact
## threshold: 0.5 for this set of metrics; using closest threshold found:
## 0.497653874378449. Run `h2o.predict` and apply your desired threshold on a
## probability column.
```

```
## Confusion Matrix (vertical: actual; across: predicted)  @ threshold = 0.497653874378449:
##        FALSE TRUE    Error     Rate
## FALSE    156   24 0.133333  =24/180
## TRUE      35  164 0.175879  =35/199
## Totals   191  188 0.155673  =59/379
```

```
print(h2o.performance(nnf))
```

```
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.111194
## RMSE:  0.3334576
## LogLoss:  0.3395033
## Mean Per-Class Error:  0.1715522
## AUC:  0.927024
## Gini:  0.854048
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        FALSE TRUE    Error     Rate
## FALSE    130   50 0.277778  =50/180
## TRUE      13  186 0.065327  =13/199
## Totals   143  236 0.166227  =63/379
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.268883 0.855172 235
## 2                       max f2  0.125889 0.905292 280
## 3                 max f0point5  0.801448 0.883534 136
## 4                 max accuracy  0.528866 0.846966 184
## 5                max precision  0.998988 1.000000   0
## 6                   max recall  0.043502 1.000000 320
## 7              max specificity  0.998988 1.000000   0
## 8             max absolute_mcc  0.528866 0.696193 184
## 9   max min_per_class_accuracy  0.470789 0.838889 195
## 10 max mean_per_class_accuracy  0.528866 0.848437 184
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

If I compute the in-sample confusion matrix directly from the in sample fitted probabilites (as I did for the logit) I get similar results.

```
yhntr = as.matrix(h2o.predict(nnf, dftrain)[,3])[,1] #in sample neural net phat
```

```
##
  |
  |                                                                      |   0%
```

```
  |
  |=================================================================| 100%
```

```r
icfn = table(dftr$y,yhntr>=.5)
icfn  #in sample
```

```
##
##          FALSE TRUE
##   FALSE    156   24
##   TRUE      36  163
```

If you like the neural net fit you can save it.

```r
h2o.saveModel(nnf, path=getwd(),force=TRUE)
```

```
## [1] "/home/rob/do/teach/18-19/ml/webpage/notes19/dnn/boston.nn_20-10"
```

And then you can read it in:

```r
fp = file.path(getwd(), "boston.nn_20-10")
if(file.exists(fp)) {
  nnfl = h2o.loadModel(fp)
}
yhntr2 = as.matrix(h2o.predict(nnfl, dftrain)[,3])[,1] #in sample neural net phat
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
plot(yhntr2,yhntr)
abline(0,1)
```