Machine Learning, Homework 6, Neural Nets

Rob McCulloch

November 21, 2019

Contents

Boston Housing with a Single Layer and R package nnet

Let's do a very simple example with single layer neural nets.

We'll do the Boston housing data with x=lstat and y =mdev so that we have one numeric x and a numeric y. We've used this classic data set a few times so we are very familiar with it.

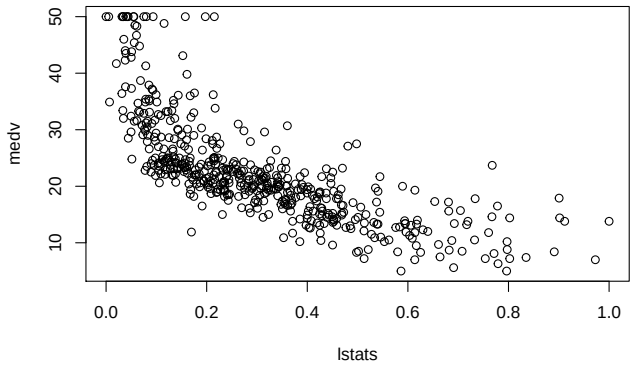
Let's get the data, pull off x and y and standardize x.

```
library(MASS) ## a library of example datasets
attach(Boston)

## standardize lstat
rg = range(Boston$lstat)
lstats = (Boston$lstat-rg[1])/(rg[2]-rg[1])

##make data frame with standardized lstat values sorted for plotting
ddf = data.frame(lstats,medv=Boston$medv)
oo = order(ddf$lstats) #order the data by x, convenient for plotting
ddf = ddf[oo,]
head(ddf)
```

```
##          lstats medv
## 162 0.000000000 50.0
## 163 0.005242826 50.0
## 41  0.006898455 34.9
## 233 0.020419426 41.7
## 193 0.031456954 36.4
## 205 0.031732892 50.0
```

And here is the familiar plot:

```
plot(ddf)
```



Let's fit a simple neural net.

One hidden layer with 5 units (neurons).

```
library(nnet)
set.seed(14)
nn1 = nnet(medv~lstats,ddf,size=5,decay=.1,linout=T,maxit=1000)
```

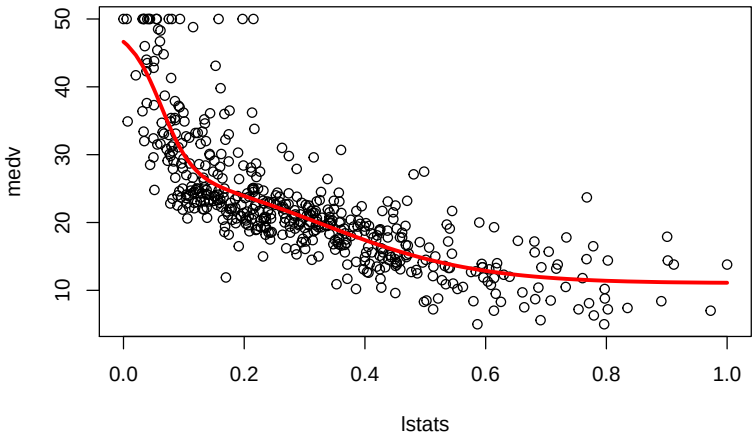
```
## # weights:  16
## initial  value 274435.143486
## iter  10 value 14655.902880
## iter  20 value 13675.210318
## iter  30 value 13618.543249
## iter  40 value 13593.167670
## iter  50 value 13548.561442
## iter  60 value 13545.520754
## iter  70 value 13544.330448
## iter  80 value 13541.583759
## iter  90 value 13540.386199
## iter 100 value 13539.604916
## iter 110 value 13536.860853
## iter 120 value 13535.643158
## iter 130 value 13535.589069
## final  value 13535.578458
## converged
```

```
summary(nn1)
```

```
## a 1-5-1 network with 16 weights
## options were - linear output units  decay=0.1
##  b->h1 i1->h1
##   1.06   0.69
##  b->h2 i1->h2
##   2.38 -38.17
##  b->h3 i1->h3
##   2.49  -7.61
##  b->h4 i1->h4
##   2.05   0.55
##  b->h5 i1->h5
##   2.53  -7.60
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o
##   4.67   3.64  21.22   9.19   3.48   8.93
```

Now let's plot the fit:

```
yhat1 = predict(nn1,ddf)
plot(ddf)
lines(ddf$lstats,yhat1,lty=1,col="red",lwd=3)
```



Notice that you understand exactly how the single layer neural fit did this !!!

Now let's fit the 5 unit neural net for a set of decay values.

Let's do this in parallel using the R parallel package. This is simple enough that we don't really need to speed it up, but we can illustrate the approach. You may want to use if for some of the more complicated model fits!

```
library(doParallel) #library for parallel computing
```

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

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

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

```
registerDoParallel()
cat("number of workers is: ",getDoParWorkers(),"\n")
```

```
## number of workers is:  4
```

```
#you could pick the number of workers with:
#  registerDoParallel(cores=num) where num is the number of workers.
```

Now we will use the function `foreach` to fit neural net models in parallel. First we set up a vector of decay values to try. Then we use `foreach` to run the neural net fits. `foreach` will return a *list*, with the i^{th} list element corresponding to the results obtained in the i^{th} loop iteration.

```
decv = c(.5,.1,.01,.005,.0025,.001,.0001,.00001)
#do a parallel loop over decay values
modsL = foreach(i=1:length(decv)) %dopar% {
   library(nnet) #I did not have to do this when I was not in Rmarkdown.
   set.seed(5*i) #I did have to to this.
   nnfit = nnet(medv~lstats,ddf,size=5,decay=decv[i],linout=T,maxit=10000)
   nnfit
}
is.list(modsL)
```

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

```
length(modsL)
```

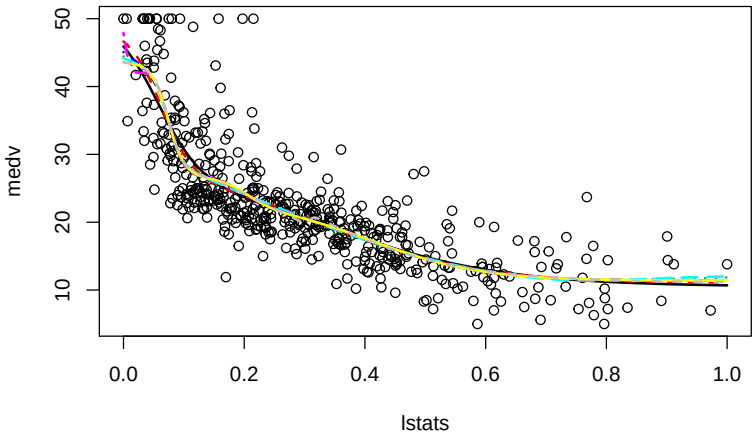
```
## [1] 8
```

The function `foreach` will launch a bunch of R processes so things like random number seeds may have to be reset for each process.

Now we can plot all the fits by looping over the list of models.

```
plot(ddf)
for(i in 1:length(modsL)) {
   yhat = predict(modsL[[i]],ddf)
```

```
   lines(ddf$lstats,yhat,col=i,lty=i,lwd=2)
}
```



Problem

- fit the neural net model with size=100 and decay=.001, plot the fits. How does it look? Try running the fit at least twice to see that it changes.
- Redo the the loop over decay values with size=100. How does it look now? Do we need 100? Will decay be more important with 100 than it was with 5 units?

Digit Recognition with R package h2o

First, let's fire up h2o.

```
print(date())
```

```
## [1] "Thu Nov 21 10:10:08 2019"
```

```
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
```

```
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmplYzOCU/h2o_root_started_from_r.out
##     /tmp/RtmplYzOCU/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 267 milliseconds
##     H2O cluster timezone:       America/Edmonton
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.26.0.2
##     H2O cluster version age:    3 months and 25 days !!!
##     H2O cluster name:           H2O_started_from_R_root_jrw534
##     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:         Amazon S3, 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 (3 months and 25 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

Now we can read in the data.

In order to make things run faster I'll down sample to just ns=10,000 observations.

```
train60D = read.csv("http://www.rob-mcculloch.org/data/mnist-train.csv")
train60D$C785 = as.factor(train60D$C785)
n = nrow(train60D)
```

```
set.seed(99)
ns = 10000
trainDS = train60D[sample(1:n,ns),]
trainS = as.h2o(trainDS,"trainS")
```

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

```
testD = read.csv("http://www.rob-mcculloch.org/data/mnist-test.csv")
testD$C785 = as.factor(testD$C785)
test = as.h2o(testD,"test")
```

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

```
x=1:784;y=785

print(ls())
```

```
##  [1] "ddf"      "decv"     "i"        "lstats"   "modsL"    "n"
##  [7] "nn1"      "ns"       "oo"       "rg"       "test"     "testD"
## [13] "train60D" "trainDS"  "trainS"   "x"        "y"        "yhat"
## [19] "yhat1"
```

```
print(h2o.ls())
```

```
##      key
## 1   test
## 2 trainS
```

Let's run `h2o.deeplearning` at settings similar to the ones that were found to work in the lecture notes. I dropped the layer/node architecture down to (50,50) so it would run faster. On my laptap in took about 90 seconds to run the one below.
I don't know how long it will take on your machine.

```
fp = file.path("./files","mDNNdrop")
if(file.exists(fp)) {
   mDNNdrop = h2o.loadModel(fp)
} else {
   tm = system.time({
   mDNNdrop = h2o.deeplearning(x,y,training_frame = trainS,
            hidden=c(50,50),
            activation="TanhWithDropout",
            hidden_dropout_ratios=c(.1,.1),
            l1=1e-4,
            epochs=2000,
            model_id="mDNNdrop",
           validation_frame=test)
   })
}
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Dropping bad and constant columns: [C646, C645, C644, C365, C760, C51, C53, C52
```

```
##
  |
  |                                                                      |   0%
  |
  |                                                                      |   1%
  |
  |=                                                                     |   1%
  |
  |=                                                                     |   2%
  |
  |==                                                                    |   2%
  |
  |==                                                                    |   3%
  |
  |==                                                                    |   4%
  |
  |===                                                                   |   4%
  |
  |======================================================================| 100%
```

```
cat("the time is: ",tm,"\n")
```

```
## the time is:  0.833 0.019 95.668 0 0
```

```
print(h2o.confusionMatrix(mDNNdrop,valid=TRUE))
```

```
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           0    1    2   3   4   5   6    7   8    9  Error           Rate
## 0       960    0    1   1   0   5   9    3   1    0 0.0204 =     20 / 980
## 1         0 1116    2   3   0   2   3    2   7    0 0.0167 =   19 / 1,135
## 2        15    3  961   6   3   2  15   12  10    5 0.0688 =   71 / 1,032
## 3         2    2   20 910   0  37   4   15  17    3 0.0990 =  100 / 1,010
## 4         1    0    6   1 903   1  15    3   4   48 0.0804 =     79 / 982
## 5        10    2    1  20   9 806  10    9  18    7 0.0964 =     86 / 892
## 6        11    4    6   0  11   8 914    0   3    1 0.0459 =     44 / 958
## 7         2    8   23   9   7   1   0  951   2   25 0.0749 =   77 / 1,028
## 8         6    7   15  12   7  11   4    9 885   18 0.0914 =     89 / 974
## 9         7    7    2   7  27   7   1   10   2  939 0.0694 =   70 / 1,009
## Totals 1014 1149 1037 969 967 880 975 1014 949 1046 0.0655 = 655 / 10,000
```

```
missclass = h2o.performance(mDNNdrop,valid=TRUE)@metrics$mean_per_class_error
cat("the mean per class error is: ",missclass,"\n")
```

```
## the mean per class error is:  0.06634004
```

```
## if you like it, keep it
#h2o.saveModel(mDNNdrop,path="./files")
print(date())
```

```
## [1] "Thu Nov 21 10:13:21 2019"
```

Problem

- I always used dropout. Is that a good idea? Change the settings to not use dropout. Is it worse or better? Do a couple of runs.
- look at ?h2o.deeplearning. Pick another option and try changing it to see if you can improve the prediction.