STAT 8330 FALL 2015 ASSIGNMENT 6

Peng Shao

October 25, 2015

(Most codes and plots are listed in the appendices)

```
time13 - time1
```

```
## Time difference of 12.28444 mins
```

▶ **Exercises 1.** **Solution.**

I fit most of models except DBN in this homework by using `train()` function in `caret` package, because it can use multiple cores to execute high performance computation, and it can easily realize cross valitadion and model comparison.

For this problem, I refit the `Carseats` dataset using gradient boosting tree, bagging tree, random forest and single-layer neural network with 10 fold cross validation. The performances of models based on training datasets are

```
apply(resamps.Carseats$values[,-1], 2, mean)
```

```
##                  ANN~RMSE              ANN~Rsquared
##                 1.0375365                 0.8691906
##         randomForest~RMSE     randomForest~Rsquared
##                 1.5568655                 0.7183086
##     GradientBoosting~RMSE GradientBoosting~Rsquared
##                 1.2410104                 0.8204771
##           BaggedCART~RMSE       BaggedCART~Rsquared
##                 1.7564050                 0.6341670
```

There is no doubt that the neural network should be the best model. The corresponding test MSEs are

```
mse.Carseats
```

```
##              ANN     randomForest GradientBoosting       BaggedCART
##         1.118744         2.685115         1.501975         3.313617
```

For neural network, I search the tuning parameters from

```
nnet.grid <- expand.grid(.decay = c(0.5, 0.1, 0.005, 0.001),
                         .size = 1:10)
```

and the best hyper parameters of the neural network are

```
nnet.fit.Carseats$bestTune
```

```
##   size decay
## 1    1 0.001
```

Then I try the `nnet()` function in `nnet` package to refit the net, using the tuning parameters above. The test MSE of the refitting neural is

```
library(nnet)
nnet.fit.Carseats.2 <- nnet(Sales ~ . , data = Carseats.train,
                            size=1, decay=0.001, linout = TRUE, trace = FALSE)
nnet.pred.Carseats.2 <- predict(nnet.fit.Carseats.2, Carseats.test, type="raw")
rmse <- mean((nnet.pred.Carseats.2 - Carseats.test$Sales)^2)
rmse
```

```
## [1] 7.813903
```

This is a little different from previouse model, which may indicate overfitting problems, since the dataset is not so "large".

▶ **Exercises 2. Solution.**

The method for this problem is identical to the previous problem. The performances of the four models based on training datasets are

```r
summary(resamps.pima)
```

```
##
## Call:
## summary.resamples(object = resamps.pima)
##
## Models: ANN, randomForest, GradientBoosting, BaggedCART
## Number of resamples: 10
##
## Accuracy
##                    Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## ANN              0.6316  0.7000 0.7560 0.7635  0.7974 0.9524    0
## randomForest     0.6500  0.7000 0.7250 0.7300  0.7589 0.8421    0
## GradientBoosting 0.7000  0.7125 0.7757 0.7751  0.8375 0.8500    0
## BaggedCART       0.6000  0.6882 0.7000 0.7245  0.7500 0.8500    0
##
## Kappa
##                     Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## ANN              0.18400  0.2767 0.4031 0.4442  0.5050 0.8889    0
## randomForest     0.00000  0.3029 0.3655 0.3658  0.4413 0.6174    0
## GradientBoosting 0.24050  0.3206 0.4806 0.4662  0.6138 0.6591    0
## BaggedCART       0.05882  0.2755 0.3407 0.3728  0.4514 0.6809    0
```

The bagging tree is the best model while the neural network becomes the worst one. But for the corresponding test error rates and confusion matrix of neural network are

```r
library(caret)
er.pima
```

```
##              ANN     randomForest GradientBoosting       BaggedCART
##        0.2530120        0.2259036        0.2168675        0.2289157
```

```r
confusionMatrix(ANN.pred.pima, Pima.te$type)[[2]]
```

```
##           Reference
## Prediction  No Yes
##        No  185  46
##        Yes  38  63
```

```r
confusionMatrix(ANN.pred.pima, Pima.te$type)[[3]][1]
```

```
## Accuracy
## 0.746988
```

```r
confusionMatrix(ANN.pred.pima, Pima.te$type)[[4]]
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.8295964            0.5779817            0.8008658
##       Neg Pred Value           Prevalence       Detection Rate
##            0.6237624            0.6716867            0.5572289
```

```
## Detection Prevalence    Balanced Accuracy
##            0.6957831            0.7037890
```

This time, the boosting tree is the best model while the neural network is still the worst one.

The best hyper parameters of the neural network are

```r
nnet.fit.pima$bestTune
```

```
##    size decay
## 29    9   0.1
```

Then I try the `nnet()` function in `nnet` package to refit the net, using the tuning parameters above. The confusion matrix and some associated results are

```r
library(nnet)
library(caret)
nnet.fit.pima.2 <- nnet(type ~ . , data = Pima.tr,
                        size=9, decay=0.1, trace = FALSE)
nnet.pred.pima.2 <- predict(nnet.fit.pima.2, Pima.te, type = "class")
confusionMatrix(as.factor(nnet.pred.pima.2), Pima.te$type)[[2]]
```

```
##           Reference
## Prediction  No Yes
##        No  195  74
##        Yes  28  35
```

```r
confusionMatrix(as.factor(nnet.pred.pima.2), Pima.te$type)[[3]][1]
```

```
##  Accuracy
## 0.6927711
```

```r
confusionMatrix(as.factor(nnet.pred.pima.2), Pima.te$type)[[4]]
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.8744395            0.3211009            0.7249071
##       Neg Pred Value           Prevalence       Detection Rate
##            0.5555556            0.6716867            0.5873494
## Detection Prevalence    Balanced Accuracy
##            0.8102410            0.5977702
```

▶ **Exercises 3. Solution.**

For deep belief network, there are many hyper (tuning) parameter needing to be chosen. But training a deep learning costs a lot of time, so I do not want to select hyper parameters using cross validation. I choose the method of random search. The candidate paramter space for searching is

```r
rand_numlayers <- sample(2:5, 1)
rand_hidden <- c(sample(10:50, rand_numlayers, T))
rand_dropout <- runif(1, 0, 0.6)
rand_learningrate <- runif(1, 0.6, 1)
dnn.fit.dig <- dbn.dnn.train(inputs.train,
                             target.train,
                             activationfun = "sigm",
                             hidden = rand_hidden,
                             output = "linear",
                             hidden_dropout = 0,
                             learningrate = rand_learningrate,
                             visible_dropout = rand_input_dropout)
```

There are the number of layers, the number of neurons in each layer, the ratio of hidden layer dropout, and the leaning rate. Here since we have very few features, I will let the input dropout be 0. Besides the hyper parameters, there are still some other options; the activation function is sigmoid function and the output function is linear function.

Then the MSE acquired from `deepnet` package is

```
dnn.mse.Carseats
```

```
## [1] 8.72375
```

and the MSE acquired from `darch` package is

```
darch.mse.Carseats
```

```
## [1] 7.830915
```

Actually, there exists some critical problem(s), which would make the prediction become identical, but I do not figure out where the problem(s) is(are). So I have to try to use `h2o` package to refit the deep network, to evaluating the deep learning methos. The MSE acquired from `h2o` is

```
h2o.mse.Carseats
```

```
## [1] 7.170961
```

Good news is, this time the predictions are not same; but bad news is, it does not improve the MSE so much.

▶ **Exercises 4. Solution.**

Like before, the method for this problem is identical to problem 3.. The only thing to do is to change the linear output funciton into softmax function.

Then the error rate acquired from `deepnet` package is

```
dnn.er.pima
```

```
## [1] 0.6716867
```

and the error rate acquired from `darch` package is

```
darch.er.pima
```

```
## [1] 0.6716867
```

The AUC acquired from `h2o` is

```
h2o.auc.pima
```

```
## [1] 0.7803513
```

▶ **Exercises 5. Solution.** (a). The code is listed in appendices, the error comfusion matrix of random forest are

```
error_rate <- 1 - confusionMatrix(rf.pred.mnist, dig_test$V785)$overall[1]
names(error_rate) <- "Error_Rate"
error_rate
```

```
## Error_Rate
##     0.0528
```

```
confusionMatrix(rf.pred.mnist, dig_test$V785)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0  967    0    9    3    1    8   13    2    6   10
##          1    1 1121    1    1    1    7    4   10    1    7
##          2    2    3  968   19    3    1    1   25    8    2
##          3    0    2    8  935    0   13    0    3   12   13
##          4    0    1   13    1  934    4    8    4    8   22
##          5    3    1    3   22    0  835    7    0   10    8
##          6    4    4    8    0    8   13  923    0   10    0
##          7    1    1   12   13    1    4    0  956    4    7
##          8    2    2    8    9    3    4    2    4  897    4
##          9    0    0    2    7   31    3    0   24   18  936
##
## Overall Statistics
##
##                Accuracy : 0.9472
##                  95% CI : (0.9426, 0.9515)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9413
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9867   0.9877   0.9380   0.9257   0.9511   0.9361
## Specificity            0.9942   0.9963   0.9929   0.9943   0.9932   0.9941
## Pos Pred Value         0.9490   0.9714   0.9380   0.9483   0.9387   0.9393
## Neg Pred Value         0.9986   0.9984   0.9929   0.9917   0.9947   0.9937
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Detection Rate         0.0967   0.1121   0.0968   0.0935   0.0934   0.0835
## Detection Prevalence   0.1019   0.1154   0.1032   0.0986   0.0995   0.0889
## Balanced Accuracy      0.9905   0.9920   0.9654   0.9600   0.9722   0.9651
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9635   0.9300   0.9209   0.9277
## Specificity            0.9948   0.9952   0.9958   0.9905
## Pos Pred Value         0.9515   0.9570   0.9594   0.9167
## Neg Pred Value         0.9961   0.9920   0.9915   0.9919
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0923   0.0956   0.0897   0.0936
## Detection Prevalence   0.0970   0.0999   0.0935   0.1021
## Balanced Accuracy      0.9791   0.9626   0.9584   0.9591
```

(b). The code is listed in appendices, but I cannot fit the model since every time I run this snippet of code, the Rstudion will terminate due to "no responding".

(c). Then the error rate acquired from `deepnet` package is

```
dnn.er.dig
```

```
## [1] 0.8968
```

and the error rate acquired from `darch` package is

```
darch.er.dig
```

```
## [1] 0.8971125
```

The error rate acquired from `h2o` is

```
h2o.er.mnist
```

```
## [1] 0.3389
```

APPENDICES

Code

```
setwd("~/Documents/git/DataAnalysis3")
rm(list = ls())
library(nnet)
library(darch)
library(deepnet)
library(caret)
library(doMC)
library(MASS)
library(h2o)
library(gbm)
library(plyr)
library(randomForest)
library(e1071)
library(ipred)
library(party)
library(mboost)
library(bst)
library(ff)
library(ISLR)
doMC::registerDoMC(cores=4)
data("Carseats")
data("Pima.tr")
data("Pima.te")
flist <- list("dig_test.RData", "dig_train.RData")
load(flist[[1]])
load(flist[[2]])
rm(flist)
dig_train <- data.frame(dig_train)
dig_train.raw <- dig_train
dig_test.raw <- dig_test
localH2O <- h2o.init(ip = "localhost", port = 54321, startH2O = TRUE)
set.seed(5262562)
data(Carseats)
trainIndex <- createDataPartition(Carseats$Sales, p=.75, list=F)
Carseats.train <- Carseats[trainIndex, ]
Carseats.test <- Carseats[-trainIndex, ]
ctrl <- trainControl(method = "cv",
                     number = 10)

# 1.
time1 <- Sys.time()
nnet.grid <- expand.grid(.decay = c(0.5, 0.1, 0.005, 0.001),
                         .size = 1:10)
```

```
nnet.fit.Carseats <- train(Sales ~ .,
                           data = Carseats.train,
                           method = "nnet",
                           maxit = 1000,
                           trControl = ctrl,
                           tuneGrid = nnet.grid,
                           preProcess = c("center", "scale"),
                           linout = TRUE,
                           trace = FALSE)
rf.fit.Carseats <- train(Sales ~ .,
                         data = Carseats.train,
                         method = "rf",
                         maxit = 1000,
                         trControl = ctrl,
                         tuneLength = 10,
                         preProcess = c("center", "scale"))
boost.grid <- expand.grid(.n.trees = seq(0, 1000, 200),
                          .interaction.depth = 1:3,
                          .shrinkage = c(0.1, 0.01, 0.001),
                          .n.minobsinnode = c(5, 10, 20))
boost.fit.Carseats <- train(Sales ~ .,
                            data = Carseats.train,
                            method ='gbm',
                            preProc = c('center','scale'),
                            tuneGrid = boost.grid,
                            trControl = ctrl)
bagging.fit.Carseats <- train(Sales ~ .,
                              data = Carseats.train,
                              method = "treebag",
                              maxit = 1000,
                              trControl = ctrl,
                              preProcess = c("center", "scale"))
resamps.Carseats <- resamples(list(ANN = nnet.fit.Carseats,
                                   randomForest = rf.fit.Carseats,
                                   GradientBoosting = boost.fit.Carseats,
                                   BaggedCART = bagging.fit.Carseats))
ANN.pred.Carseats <- predict(nnet.fit.Carseats, newdata = Carseats.test)
rf.pred.Carseats <- predict(rf.fit.Carseats, newdata = Carseats.test)
boost.pred.Carseats <- predict(boost.fit.Carseats, newdata = Carseats.test)
bagging.pred.Carseats <- predict(bagging.fit.Carseats, newdata = Carseats.test)
pred.Carseats <-matrix(c(ANN.pred.Carseats,
                         rf.pred.Carseats,
                         boost.pred.Carseats,
                         bagging.pred.Carseats),
                       ncol = 4,
                       byrow = FALSE)
mse <- function(r){mean(r^2)}
mse.Carseats <- apply(pred.Carseats - Carseats.test$Sales, 2, mse)
names(mse.Carseats) <- c("ANN", "randomForest", "GradientBoosting", "BaggedCART")


#2
time2 <- Sys.time()
```

```
ctrl <- trainControl(method = "cv",
                     number = 10,
                     classProbs = TRUE)
nnet.grid <- expand.grid(.decay = c(0.5, 0.1, 0.005, 0.001),
                         .size = 1:10)
nnet.fit.pima <- train(type ~ .,
                       data = Pima.tr,
                       method = "nnet",
                       maxit = 1000,
                       trControl = ctrl,
                       tuneGrid = nnet.grid,
                       preProcess = c("center", "scale"),
                       trace = FALSE)
rf.fit.pima <- train(type ~ .,
                     data = Pima.tr,
                     method = "rf",
                     maxit = 1000,
                     trControl = ctrl,
                     tuneLength = 10,
                     preProcess = c("center", "scale"))
boost.grid <- expand.grid(.n.trees = seq(0, 1000, 200),
                          .interaction.depth = 1:3,
                          .shrinkage = c(0.1, 0.01, 0.001),
                          .n.minobsinnode = c(5, 10, 20))
boost.fit.pima <- train(type ~ .,
                        data = Pima.tr,
                        method = 'gbm',
                        preProc = c('center','scale'),
                        tuneGrid = boost.grid,
                        trControl = ctrl)
bagging.fit.pima <- train(type ~ .,
                          data = Pima.tr,
                          method = "treebag",
                          maxit = 1000,
                          trControl = ctrl,
                          preProcess = c("center", "scale"))
resamps.pima <- resamples(list(ANN = nnet.fit.pima,
                               randomForest = rf.fit.pima,
                               GradientBoosting = boost.fit.pima,
                               BaggedCART = bagging.fit.pima))
ANN.pred.pima <- predict(nnet.fit.pima, newdata = Pima.te)
rf.pred.pima <- predict(rf.fit.pima, newdata = Pima.te)
boost.pred.pima <- predict(boost.fit.pima, newdata = Pima.te)
bagging.pred.pima <- predict(bagging.fit.pima, newdata = Pima.te)
pred.pima <-matrix(c(ANN.pred.pima,
                     rf.pred.pima,
                     boost.pred.pima,
                     bagging.pred.pima),
                   ncol = 4,
                   byrow = FALSE)
er <- function(wrong){mean(abs(wrong))}
er.pima <- apply(pred.pima - as.numeric(Pima.te$type), 2, er)
names(er.pima) <- c("ANN", "randomForest", "GradientBoosting", "BaggedCART")
```

```r
time3 <- Sys.time()


#3.
#deepnet
inputs.train <- model.matrix(Sales ~ ., data = Carseats.train)
inputs.test <- model.matrix(Sales ~ ., data = Carseats.test)
models <- vector("list", 5)
models.mse <- c()
for (i in 1:5){
        rand_numlayers <- sample(2:5, 1)
        rand_hidden <- c(sample(10:50, rand_numlayers, T))
        rand_dropout <- runif(1, 0, 0.6)
        rand_learningrate <- runif(1, 0.6, 1)
        dnn.fit.Carseats <- dbn.dnn.train(inputs.train,
                                          Carseats.train[, 1],
                                          hidden = rand_hidden,
                                          activationfun = "sigm",
                                          output = "linear",
                                          hidden_dropout = rand_dropout,
                                          learningrate = rand_learningrate,
                                          visible_dropout = 0)
        dnn.pred.Carseats <- nn.predict(dnn.fit.Carseats,
                                        inputs.test)
        dnn.mse.Carseats <- mean((dnn.pred.Carseats - Carseats.test[, 1])^2)
        models[[i]] <- dnn.fit.Carseats
        models.mse <- c(models.mse, dnn.mse.Carseats)
}
best.err <- models.mse[1]
best.model <- models[[1]]
for (i in 1:length(models)) {
        err <- models.mse[i]
        if (err < best.err) {
                best.err <- err
                best.model <- models[[i]]
        }
}
dnn.fit.Carseats <- best.model
dnn.mse.Carseats <- best.err
dnn.pred.Carseats <- nn.predict(dnn.fit.Carseats,
                                inputs.test)
time4 <- Sys.time()

# darch
darch.fit.Carseats <- newDArch(dnn.fit.Carseats$size,
                               batchSize=4,
                               ff=F)
darch.fit.Carseats <- preTrainDArch(darch.fit.Carseats,
                                    inputs.train,
                                    maxEpoch = 10,
                                    numCD = 4)
layers <- getLayers(darch.fit.Carseats)
layers[[length(layers)]][[2]] <- linearUnitDerivative
```

```
for(i in (length(layers) - 1):1){
        layers[[i]][[2]] <- sigmoidUnitDerivative
}
setLayers(darch.fit.Carseats) <- layers
rm(layers)
setFineTuneFunction(darch.fit.Carseats) <- rpropagation
darch.fit.Carseats <- fineTuneDArch(darch.fit.Carseats,
                                  trainData = inputs.train,
                                  targetData = matrix(Carseats.train$Sales),
                                  maxEpoch = 10,
                                  isBin = T)

# Running the darch
darch.pred.Carseats <-
        darch.pred.Carseats <-
        getExecuteFunction(darch.fit.Carseats)(darch.fit.Carseats,inputs.test)
outputs2 <- getExecOutputs(darch.pred.Carseats)
darch.mse.Carseats <- mse(outputs2[[length(outputs2)]]-Carseats.test$Sales)
time5 <- Sys.time()

# h2o
Carseats.train.h2o <- as.h2o(localH2O, model.matrix(~ . - 1 , data = Carseats.train))
Carseats.test.h2o <- as.h2o(localH2O, model.matrix(~ . - 1 , data = Carseats.test))
models <- c()
for (i in 1:10) {
        rand_activation <- c("TanhWithDropout", "RectifierWithDropout")[sample(1:2,1)]
        rand_numlayers <- sample(2:5,1)
        rand_hidden <- c(sample(10:50,rand_numlayers,T))
        rand_l1 <- runif(1, 0, 1e-3)
        rand_l2 <- runif(1, 0, 1e-3)
        rand_dropout <- c(runif(rand_numlayers, 0, 0.6))
        rand_input_dropout <- runif(1, 0, 0.5)
        dlmodel <- h2o.deeplearning(x = 2:13,
                                  y = 1,
                                  training_frame = Carseats.train.h2o,
                                  validation_frame = Carseats.test.h2o,
                                  epochs = 0.1,
                                  activation = rand_activation,
                                  hidden = rand_hidden,
                                  l1 = rand_l1,
                                  l2 = rand_l2,
                                  input_dropout_ratio = 0,
                                  hidden_dropout_ratios = rand_dropout)
        models <- c(models, dlmodel)
}
best_mse <- models[[1]]@model$validation_metrics@metrics$MSE #best model from grid search above
for (i in 1:length(models)) {
        mse.h2o <- models[[i]]@model$validation_metrics@metrics$MSE
        if (mse.h2o < best_mse) {
                best_mse <- mse.h2o
                best_model <- models[[i]]
        }
}
```

```
h2o.fit.Carseats <- best_model
h2o.mse.Carseats <- best_mse

time6 <- Sys.time()

#4.
#deepnet
inputs.train <- model.matrix(type ~ ., data = Pima.tr)
inputs.test <- model.matrix(type ~ ., data = Pima.te)
target.train <- as.numeric(Pima.tr$type) - 1
target.test <- as.numeric(Pima.te$type) - 1
models <- vector("list", 5)
models.er <- c()
for (i in 1:5){
        rand_numlayers <- sample(2:5, 1)
        rand_dropout <- runif(1, 0, 0.6)
        rand_hidden <- c(sample(10:50, rand_numlayers, T))
        rand_learningrate <- runif(1, 0.6, 1)
        dnn.fit.pima <- dbn.dnn.train(inputs.train,
                                      target.train,
                                      activationfun = "sigm",
                                      hidden = rand_hidden,
                                      output = "softmax",
                                      hidden_dropout = 0,
                                      learningrate = rand_learningrate,
                                      visible_dropout = 0)
        dnn.pred.pima <- nn.predict(dnn.fit.pima,
                                    inputs.test)
        dnn.er.pima <- er(dnn.pred.pima - target.test)
        models[[i]] <- dnn.fit.pima
        models.er <- c(models.er, dnn.er.pima)
}
best.err <- models.er[1]
best.model <- models[[1]]
for (i in 1:length(models)) {
        err <- models.er[i]
        if (err < best.err) {
                best.err <- err
                best.model <- models[[i]]
        }
}
dnn.fit.pima <- best.model
dnn.er.pima <- best.err
dnn.pred.pima <- nn.predict(dnn.fit.pima,
                            inputs.test)
time7 <- Sys.time()

# darch
darch.fit.pima <- newDArch(dnn.fit.pima$size,
                           batchSize=4,
                           ff=F)
darch.fit.pima <- preTrainDArch(darch.fit.pima,
                                inputs.train,
```

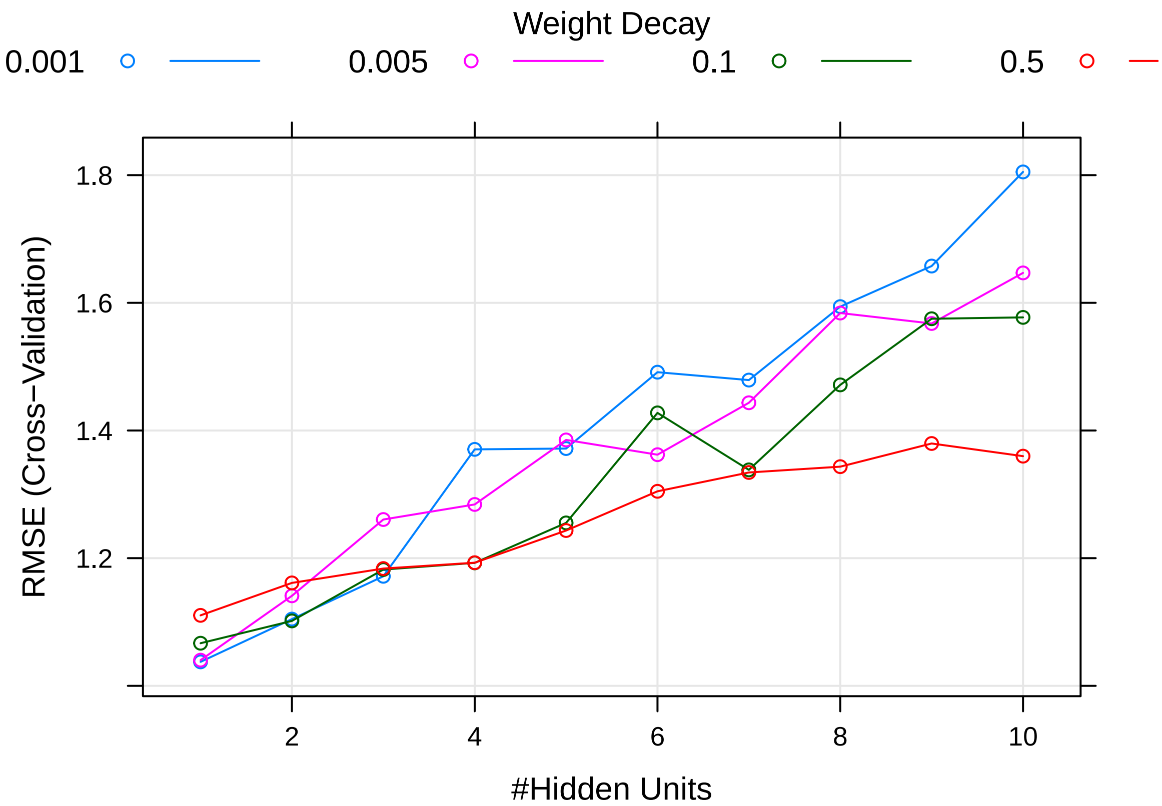
```
                                maxEpoch = 10,
                                numCD = 4)
layers <- getLayers(darch.fit.pima)
layers[[length(layers)]][[2]] <- softmaxUnitDerivative
for(i in (length(layers) - 1):1){
        layers[[i]][[2]] <- sigmoidUnitDerivative
}
setLayers(darch.fit.pima) <- layers
rm(layers)
setFineTuneFunction(darch.fit.pima) <- rpropagation
darch.fit.pima <- fineTuneDArch(darch.fit.pima,
                                trainData = inputs.train,
                                targetData = matrix(target.train),
                                maxEpoch = 10,
                                isBin = T)
# Running the darch
darch.pred.pima <-
        darch.pred.pima <-
        getExecuteFunction(darch.fit.pima)(darch.fit.pima, inputs.test)
outputs2 <- getExecOutputs(darch.pred.pima)
darch.er.pima <- er(outputs2[[length(outputs2)]]-target.test)
time8 <- Sys.time()

#h2o
pima.train.h2o <- as.h2o(localH2O, Pima.tr)
pima.test.h2o <- as.h2o(localH2O, Pima.te)
models <- c()
for (i in 1:10) {
        rand_activation <- c("TanhWithDropout", "RectifierWithDropout")[sample(1:2,1)]
        rand_numlayers <- sample(2:5,1)
        rand_hidden <- c(sample(10:50,rand_numlayers,T))
        rand_l1 <- runif(1, 0, 1e-3)
        rand_l2 <- runif(1, 0, 1e-3)
        rand_dropout <- c(runif(rand_numlayers, 0, 0.6))
        rand_input_dropout <- runif(1, 0, 0.5)
        dlmodel <- h2o.deeplearning(x = 1:7,
                                    y = 8,
                                    training_frame = pima.train.h2o,
                                    validation_frame = pima.test.h2o,
                                    epochs = 0.1,
                                    activation = rand_activation,
                                    hidden = rand_hidden,
                                    l1 = rand_l1,
                                    l2 = rand_l2,
                                    input_dropout_ratio = 0,
                                    hidden_dropout_ratios = rand_dropout)
        models <- c(models, dlmodel)
}
best_auc <- models[[1]]@model$validation_metrics@metrics$AUC #best model from grid search above
for (i in 1:length(models)) {
        auc <- models[[i]]@model$validation_metrics@metrics$AUC
        if (auc > best_auc) {
                best_auc <- auc
```

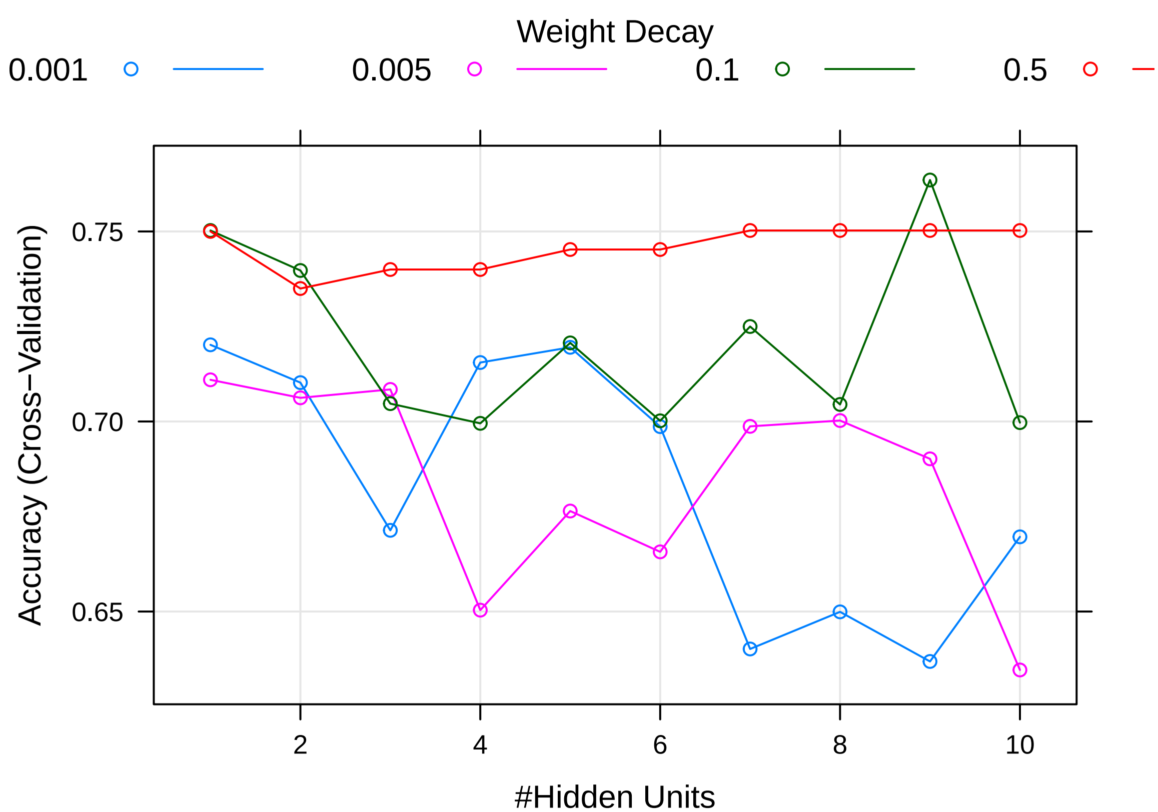
```
                best_model <- models[[i]]
        }
}
h2o.fit.pima <- best_model
h2o.auc.pima <- best_auc
time9 <- Sys.time()







# 5
(a).
dig_train[, 785] <- as.factor(dig_train[, 785])
colnames(dig_train) <- paste("V", 1:785, sep = '')
colnames(dig_test) <- paste("V", 1:785, sep = '')
rf.fit.mnist <- randomForest(V785 ~ . , data = dig_train)
rf.pred.mnist <- predict(rf.fit.mnist, dig_test)
time10 <- Sys.time()

(b).
clsresp.train <- class.ind(dig_train$V785)
nnet.fit.mnist = nnet(dig_train[,-785], clsresp.train,
                      size = 50,
                      decay = 0.2,
                      softmax = TRUE,
                      entropy = TRUE,
                      MaxNWts = 1e6)

(c).
#deepnet

inputs.train <- model.matrix(V785 ~ . - 1, data = dig_train)
inputs.test <- model.matrix(V785 ~ . - 1, data = dig_test)
target.train <- class.ind(dig_train$V785)
target.test <- class.ind(dig_test$V785)

models <- vector("list", 5)
models.er <- c()
trans <- function(output){
        ind <- which.max(output)
        output[ind] <- 1
        output[-ind] <- 0
        return(output)
}
for (i in 1:5){
        rand_numlayers <- sample(1:3, 1)
        rand_hidden <- c(sample(10:50,rand_numlayers,T))
        rand_dropout <- runif(1, 0, 0.6)
        rand_input_dropout <- runif(1, 0, 0.5)
        rand_learningrate <- runif(1, 0.6, 1)
```

```
        dnn.fit.dig <- dbn.dnn.train(inputs.train,
                                     target.train,
                                     activationfun = "sigm",
                                     hidden = rand_hidden,
                                     output = "softmax",
                                     hidden_dropout = rand_dropout,
                                     learningrate = rand_learningrate,
                                     visible_dropout = rand_input_dropout)
        dnn.pred.dig <- nn.predict(dnn.fit.dig,
                                   inputs.test)
        dnn.pred.dig <- t(apply(dnn.pred.dig, 1, trans))
        dnn.er.dig <- (sum(abs(dnn.pred.dig - target.test))/2)/nrow(target.test)
        models[[i]] <- dnn.fit.dig
        models.er <- c(models.er, dnn.er.dig)
}
best.err <- models.er[1]
best.model <- models[[1]]
for (i in 1:length(models)) {
        err <- models.er[i]
        if (err < best.err) {
                best.err <- err
                best.model <- models[[i]]
        }
}
dnn.fit.dig <- best.model
dnn.er.dig <- best.err
dnn.pred.dig <- nn.predict(dnn.fit.dig,
                           inputs.test)
time11 <- Sys.time()

# darch
darch.fit.dig <- newDArch(dnn.fit.dig$size,
                          batchSize=4,
                          ff=F)
darch.fit.dig <- preTrainDArch(darch.fit.dig,
                               inputs.train,
                               maxEpoch = 10,
                               numCD = 4)
layers <- getLayers(darch.fit.dig)
layers[[length(layers)]][[2]] <- softmaxUnitDerivative
for(i in (length(layers) - 1):1){
        layers[[i]][[2]] <- sigmoidUnitDerivative
}
setLayers(darch.fit.dig) <- layers
rm(layers)
setFineTuneFunction(darch.fit.dig) <- rpropagation
darch.fit.dig <- fineTuneDArch(darch.fit.dig,
                               trainData = inputs.train,
                               targetData = target.train,
                               maxEpoch = 10,
                               isBin = T)
# Running the darch
darch.pred.dig <-
```

```
        darch.pred.dig <-
        getExecuteFunction(darch.fit.dig)(darch.fit.dig, inputs.test)
outputs2 <- getExecOutputs(darch.pred.dig)
darch.pred.dig <- t(apply(outputs2[[length(outputs2)]], 1, trans))
darch.er.dig <- (sum(abs(dnn.pred.dig - target.test))/2)/nrow(target.test)
time12 <- Sys.time()

# H2O
mnist.train <- as.h2o(localH2O, dig_train)
mnist.test <- as.h2o(localH2O, dig_test)
models <- c()
for (i in 1:10) {
        rand_activation <- c("TanhWithDropout", "RectifierWithDropout")[sample(1:2,1)]
        rand_numlayers <- sample(2:5,1)
        rand_hidden <- c(sample(10:50,rand_numlayers,T))
        rand_l1 <- runif(1, 0, 1e-3)
        rand_l2 <- runif(1, 0, 1e-3)
        rand_dropout <- c(runif(rand_numlayers, 0, 0.6))
        rand_input_dropout <- runif(1, 0, 0.5)
        dlmodel <- h2o.deeplearning(x = 1:784,
                                    y = 785,
                                    training_frame = mnist.train,
                                    validation_frame = mnist.test,
                                    epochs = 0.1,
                                    activation = rand_activation,
                                    hidden = rand_hidden,
                                    l1 = rand_l1,
                                    l2 = rand_l2,
                                    input_dropout_ratio = rand_input_dropout,
                                    hidden_dropout_ratios = rand_dropout)
        models <- c(models, dlmodel)
}
best_err <- models[[1]]@model$validation_metrics@metrics$cm$table[11, 11] #best model from grid search
for (i in 1:length(models)) {
        err <- models[[i]]@model$validation_metrics@metrics$cm$table[11, 11]
        if (err < best_err) {
                best_err <- err
                best_model <- models[[i]]
        }
}
h2o.fit.mnist <- best_model
h2o.er.mnist <- best_err
time13 <- Sys.time()
```

Plot


Weight Decay
0.001
0.005
0.1
0.5
RMSE (Cross−Validation)
1.8
1.6
1.4
1.2
2
4
6
8
10
#Hidden Units


Weight Decay
0.001
0.005
0.1
0.5
Accuracy (Cross−Validation)
0.75
0.70
0.65
2
4
6
8
10
#Hidden Units