STAT 8330 FALL 2015 ASSIGNMENT 6

Peng Shao

October 25, 2015

(Most codes and plots are listed in the appendices)

▶ **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] 6.366908
```

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

```
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

```
library(caret)
```

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

```
er.pima
```

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

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

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

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

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

```
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

```
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

```
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  174  49
##        Yes  49  60
```

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

```
##  Accuracy
## 0.7048193
```

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

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.7802691            0.5504587            0.7802691
##        Neg Pred Value           Prevalence       Detection Rate
##            0.5504587            0.6716867            0.5240964
## Detection Prevalence    Balanced Accuracy
##            0.6716867            0.6653639
```

► **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

```
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.

▶ **Exercises 5. Solution.**

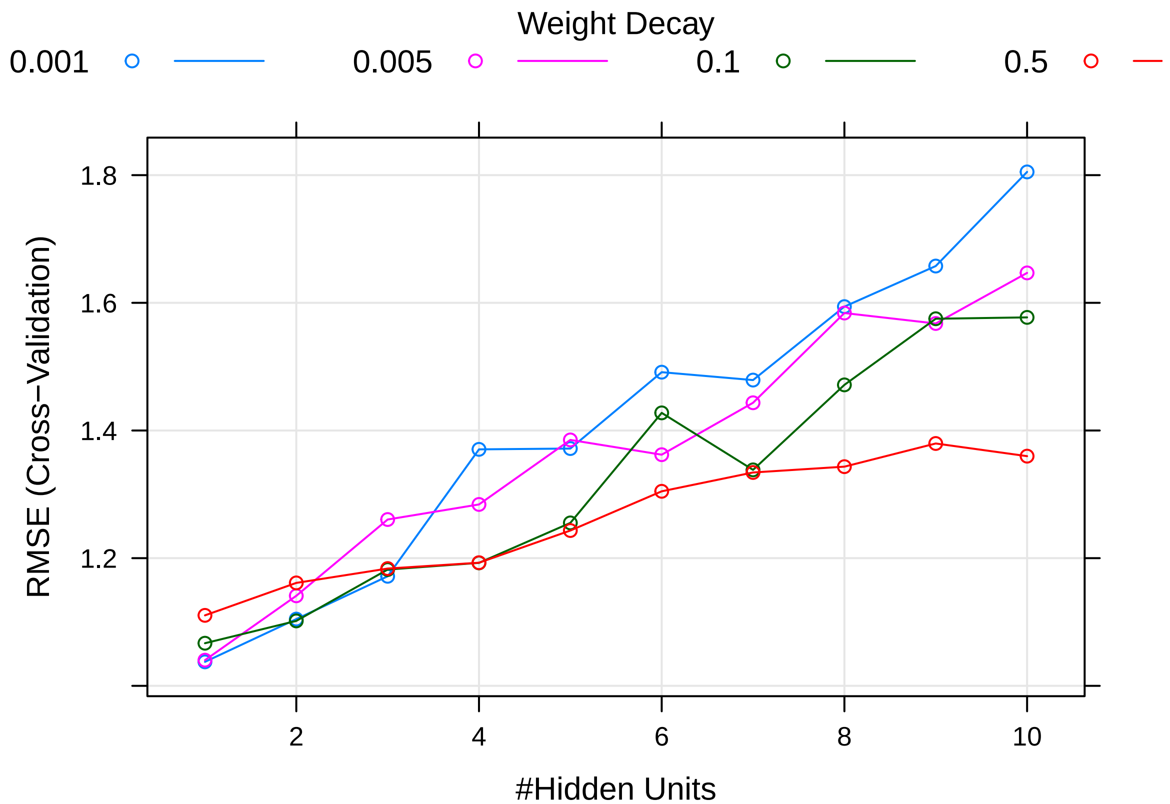
APPENDICES

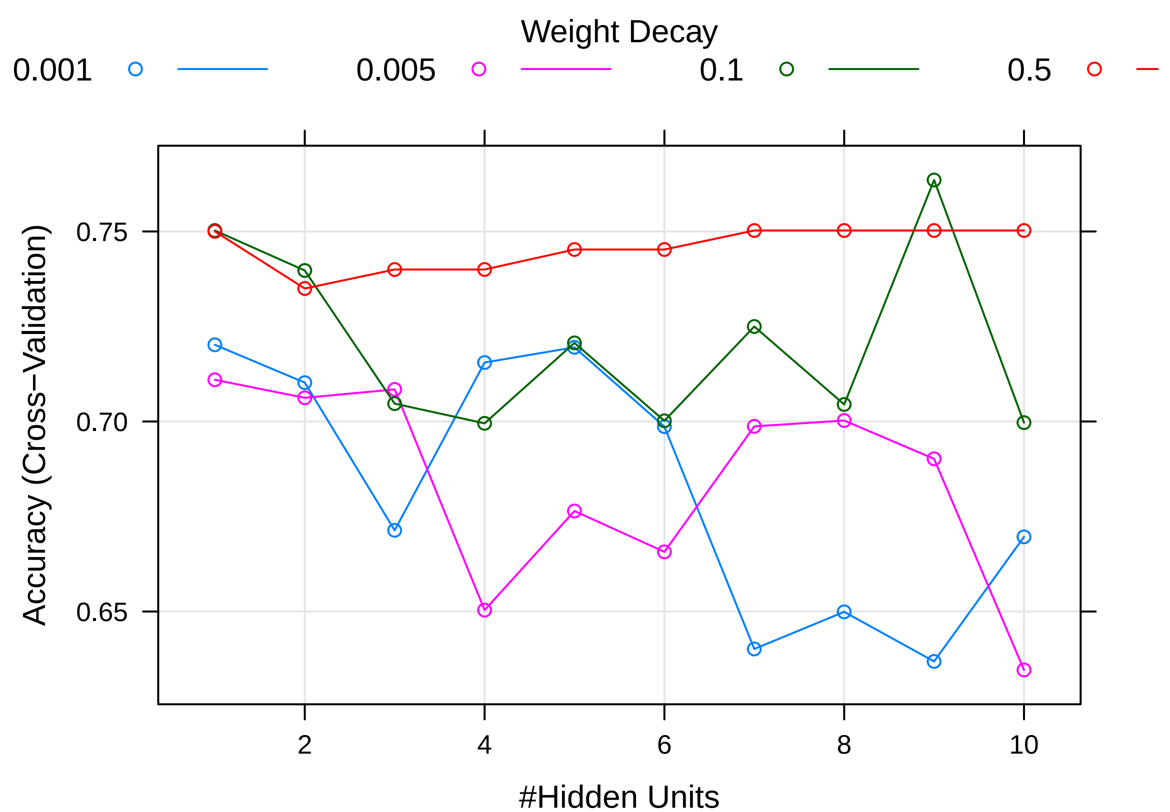
Code

```
time13-time1
```

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

Plot


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


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