Thesis Draft One

Gerhard Viljoen - VLJCHR004

12 September 2018

## Set up session for Deep Learning:

Set working directory:

#setwd("~/Documents/R")

Clear R environment objects:

rm(list=ls())

Read in JSON dump of Chris’s python dictionary and convert to three dataframes:

require(jsonlite)  
  
dat <- fromJSON("data/text.txt")  
  
zero <- dat[[1]]  
one <- dat[[2]]  
two <- dat[[3]]  
three <- dat[[4]]

Union these dataframes into one:

all.dat <- data.frame(  
 rbind(  
 zero,  
 one,  
 two,  
 three  
 )  
)

# Data Wrangling for Deep Learning:

require(dplyr)  
library(tidyr)  
  
#isolate the layer columns  
layers <- data.frame(all.dat[,c(8,12,16,20,24,28)])  
  
#choose an example that is not null  
fix <- layers[6,1]  
  
#fill it with zeros, retaining its shape  
for(i in 1:nrow(fix[[1]])){  
 for(j in 1:ncol(fix[[1]])){  
 fix[[1]][i,j] <- 0  
 }  
}  
  
#replace any null values with this matrix of zeroes  
nullToNA <- function(x) {  
 x[sapply(x, is.null)] <- fix  
 return(x)  
}  
  
layers <- as.matrix(layers)  
layers <- nullToNA(layers)  
  
layers <- cbind(all.dat[,1:3],layers)  
  
#unnest each layer individually:  
#//TODO: create a function for this  
  
#LAYER 0  
  
a <- rep(NA,264)  
  
for(i in layers$layer0){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(all.dat[,c(1:7,9:11,13:15,17:19,21:23,25:27)],a)  
  
#LAYER 1  
  
a <- rep(NA,264)  
  
for(i in layers$layer1){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 2  
  
a <- rep(NA,264)  
  
for(i in layers$layer2){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 3  
  
a <- rep(NA,264)  
  
for(i in layers$layer3){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 4  
  
a <- rep(NA,264)  
  
for(i in layers$layer4){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#LAYER 5  
  
a <- rep(NA,264)  
  
for(i in layers$layer5){  
 add <- as.vector(unlist(i))  
 a <- rbind(a,add)  
}  
  
a <- a[-1,]  
  
layer0 <- cbind(layer0,a)  
  
#make all missing values 0  
layer0[is.na(layer0)] <- 0  
  
my.dat <- layer0[,4:100]  
  
#make the outcome variable categorical  
my.dat$pdgCode <- as.factor(my.dat$pdgCode)  
  
#save the manipulated dataset and clear the R environment  
save(my.dat,file="NNdata.RData")  
rm(list=ls())

Scale the wrangled data and remove unnecessary elements:

#load the manipulated data set  
load("NNdata.RData")  
#scale numerical predictor variables  
my.dat[,-1] <- scale(my.dat[,-1])  
my.dat <- my.dat %>%  
 subset(pdgCode!=-11) %>%  
 subset(pdgCode!=11)

# DEEP LEARNING:

require(h2o)  
h2o.init(max\_mem\_size = "28G",nthreads = -1)

##   
## H2O is not running yet, starting it now...  
##   
## Note: In case of errors look at the following log files:  
## /tmp/Rtmptu2R5K/h2o\_metamorphica\_started\_from\_r.out  
## /tmp/Rtmptu2R5K/h2o\_metamorphica\_started\_from\_r.err  
##   
##   
## Starting H2O JVM and connecting: . Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 1 seconds 308 milliseconds   
## H2O cluster timezone: Africa/Johannesburg   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.20.0.8   
## H2O cluster version age: 13 days   
## H2O cluster name: H2O\_started\_from\_R\_metamorphica\_dte617   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 24.89 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 3.4.4 (2018-03-15)

h2o.removeAll()

## [1] 0

#h2o.no\_progress()

Upload data to H2O cluster, split into training (60%), validation (20%) and test (20%) sets:

dat.hex <- as.h2o(my.dat,"dat.hex")

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

splitz <- h2o.splitFrame(dat.hex,ratios=c(0.6,0.2),  
 destination\_frames = c("train.hex","valid.hex","test.hex"))  
  
train.hex <- splitz[[1]]  
valid.hex <- splitz[[2]]  
test.hex <- splitz[[3]]

Build first neural network:

* 2 hidden layers of 200 neurons each
* 100 epochs
* 10-fold cross validation

nn\_1 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_1",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = c(200,200),  
 nfolds=10,  
 standardize = F,  
 epochs=100,  
 fast\_mode = F,  
 sparse = T  
)

##   
 |   
 | | 0%  
 |   
 |===== | 8%  
 |   
 |====== | 10%  
 |   
 |======== | 12%  
 |   
 |=========== | 17%  
 |   
 |============= | 20%  
 |   
 |============== | 21%  
 |   
 |================== | 27%  
 |   
 |=================== | 29%  
 |   
 |====================== | 33%  
 |   
 |======================== | 37%  
 |   
 |========================== | 41%  
 |   
 |============================= | 45%  
 |   
 |=============================== | 47%  
 |   
 |================================ | 50%  
 |   
 |================================== | 52%  
 |   
 |==================================== | 56%  
 |   
 |====================================== | 59%  
 |   
 |========================================== | 65%  
 |   
 |============================================= | 70%  
 |   
 |=============================================== | 72%  
 |   
 |================================================= | 76%  
 |   
 |=================================================== | 79%  
 |   
 |====================================================== | 83%  
 |   
 |======================================================== | 87%  
 |   
 |========================================================== | 90%  
 |   
 |=================================================================| 100%

Build a differenct architecture:

* One hidden layer of 500 neurons
* 100 epochs
* 10-fold cross validation

nn\_2 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_2",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = 500,  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=100  
)

##   
 |   
 | | 0%  
 |   
 |= | 1%  
 |   
 |=== | 5%  
 |   
 |======= | 11%  
 |   
 |========== | 15%  
 |   
 |============= | 20%  
 |   
 |================ | 24%  
 |   
 |================== | 28%  
 |   
 |===================== | 32%  
 |   
 |======================= | 35%  
 |   
 |========================= | 39%  
 |   
 |============================ | 44%  
 |   
 |=============================== | 47%  
 |   
 |=================================== | 54%  
 |   
 |======================================= | 59%  
 |   
 |============================================ | 67%  
 |   
 |=============================================== | 72%  
 |   
 |================================================== | 77%  
 |   
 |===================================================== | 81%  
 |   
 |======================================================= | 84%  
 |   
 |=========================================================== | 90%  
 |   
 |============================================================ | 92%  
 |   
 |=============================================================== | 97%  
 |   
 |=================================================================| 100%

Build yet another differenct architecture:

* 10 hidden layers
* 100 epochs
* 10-fold cross validation

nn\_3 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_3",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 hidden = c(32,32,32,32,32,32,32,32,32,32),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=100  
)

##   
 |   
 | | 0%  
 |   
 |====== | 9%  
 |   
 |================= | 27%  
 |   
 |============================== | 46%  
 |   
 |======================================== | 61%  
 |   
 |================================================== | 77%  
 |   
 |=================================================================| 100%

Find most probable architecture:

h2o.mean\_per\_class\_error(nn\_1)

## [1] 0.2241379

h2o.mean\_per\_class\_error(nn\_2)

## [1] 0.2155172

h2o.mean\_per\_class\_error(nn\_3)

## [1] 0.2241379

h2o.confusionMatrix(nn\_1)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.0230669511644981:  
## -211 211 Error Rate  
## -211 32 26 0.448276 =26/58  
## 211 0 66 0.000000 =0/66  
## Totals 32 92 0.209677 =26/124

h2o.confusionMatrix(nn\_2)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.120827021588128:  
## -211 211 Error Rate  
## -211 33 25 0.431034 =25/58  
## 211 0 66 0.000000 =0/66  
## Totals 33 91 0.201613 =25/124

h2o.confusionMatrix(nn\_3)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.566688777517043:  
## -211 211 Error Rate  
## -211 32 26 0.448276 =26/58  
## 211 0 66 0.000000 =0/66  
## Totals 32 92 0.209677 =26/124

h2o.confusionMatrix(nn\_1,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.416494562377328:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 1 20 0.047619 =1/21  
## Totals 7 30 0.297297 =11/37

h2o.confusionMatrix(nn\_2,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.120827021588128:  
## -211 211 Error Rate  
## -211 4 12 0.750000 =12/16  
## 211 0 21 0.000000 =0/21  
## Totals 4 33 0.324324 =12/37

h2o.confusionMatrix(nn\_3,newdata = test.hex)

## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.566688777517043:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 0 21 0.000000 =0/21  
## Totals 6 31 0.270270 =10/37

plot(nn\_1)



plot(nn\_2)



plot(nn\_3)



# Tuning:

The above information suggests fewer hidden layers, with more neurons is more suited to this problem.

We build a fourth neural network architecture: \* 2 hidden layers of 500 neurons each \* 200 epochs

And test this with various activation functions:

## Tanh with Dropout

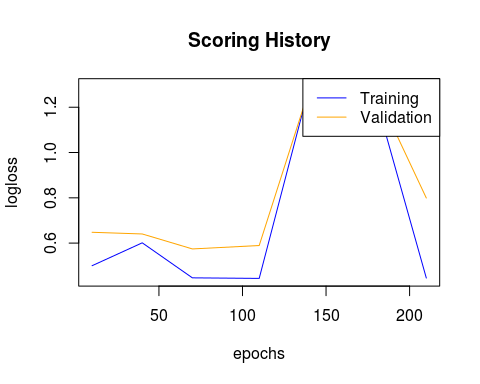
nn\_4 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_4",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "TanhWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

##   
 |   
 | | 0%  
 |   
 |= | 1%  
 |   
 |= | 2%  
 |   
 |=== | 4%  
 |   
 |=== | 5%  
 |   
 |==== | 6%  
 |   
 |===== | 7%  
 |   
 |===== | 8%  
 |   
 |====== | 9%  
 |   
 |====== | 10%  
 |   
 |======= | 10%  
 |   
 |======= | 11%  
 |   
 |======== | 12%  
 |   
 |========= | 13%  
 |   
 |========== | 15%  
 |   
 |========== | 16%  
 |   
 |=========== | 16%  
 |   
 |=========== | 17%  
 |   
 |=========== | 18%  
 |   
 |============ | 18%  
 |   
 |============ | 19%  
 |   
 |============= | 19%  
 |   
 |============= | 20%  
 |   
 |============== | 21%  
 |   
 |============== | 22%  
 |   
 |=============== | 22%  
 |   
 |=============== | 23%  
 |   
 |=============== | 24%  
 |   
 |================ | 24%  
 |   
 |================ | 25%  
 |   
 |================= | 26%  
 |   
 |================== | 27%  
 |   
 |================== | 28%  
 |   
 |=================== | 29%  
 |   
 |=================== | 30%  
 |   
 |==================== | 31%  
 |   
 |===================== | 33%  
 |   
 |====================== | 33%  
 |   
 |====================== | 34%  
 |   
 |======================= | 35%  
 |   
 |======================= | 36%  
 |   
 |======================== | 37%  
 |   
 |========================= | 38%  
 |   
 |========================= | 39%  
 |   
 |========================== | 40%  
 |   
 |=========================== | 41%  
 |   
 |=========================== | 42%  
 |   
 |============================ | 43%  
 |   
 |============================= | 44%  
 |   
 |============================= | 45%  
 |   
 |============================== | 46%  
 |   
 |============================== | 47%  
 |   
 |=============================== | 47%  
 |   
 |=============================== | 48%  
 |   
 |================================ | 49%  
 |   
 |================================= | 50%  
 |   
 |================================= | 51%  
 |   
 |================================== | 52%  
 |   
 |=================================== | 54%  
 |   
 |==================================== | 55%  
 |   
 |===================================== | 56%  
 |   
 |===================================== | 57%  
 |   
 |====================================== | 58%  
 |   
 |====================================== | 59%  
 |   
 |======================================= | 59%  
 |   
 |======================================= | 60%  
 |   
 |======================================== | 61%  
 |   
 |========================================= | 63%  
 |   
 |========================================== | 64%  
 |   
 |========================================== | 65%  
 |   
 |=========================================== | 66%  
 |   
 |============================================ | 67%  
 |   
 |============================================ | 68%  
 |   
 |============================================= | 69%  
 |   
 |============================================= | 70%  
 |   
 |============================================== | 70%  
 |   
 |============================================== | 71%  
 |   
 |=============================================== | 72%  
 |   
 |================================================ | 73%  
 |   
 |================================================ | 74%  
 |   
 |================================================= | 75%  
 |   
 |================================================= | 76%  
 |   
 |================================================== | 77%  
 |   
 |=================================================== | 78%  
 |   
 |==================================================== | 80%  
 |   
 |===================================================== | 81%  
 |   
 |===================================================== | 82%  
 |   
 |====================================================== | 83%  
 |   
 |======================================================= | 85%  
 |   
 |======================================================== | 86%  
 |   
 |======================================================== | 87%  
 |   
 |========================================================= | 87%  
 |   
 |========================================================= | 88%  
 |   
 |========================================================== | 89%  
 |   
 |========================================================== | 90%  
 |   
 |=========================================================== | 90%  
 |   
 |=========================================================== | 91%  
 |   
 |============================================================ | 92%  
 |   
 |============================================================ | 93%  
 |   
 |============================================================= | 94%  
 |   
 |============================================================== | 95%  
 |   
 |============================================================== | 96%  
 |   
 |=============================================================== | 96%  
 |   
 |=============================================================== | 97%  
 |   
 |================================================================ | 99%  
 |   
 |=================================================================| 100%

h2o.performance(nn\_4,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.3263933  
## RMSE: 0.5713084  
## LogLoss: 1.019133  
## Mean Per-Class Error: 0.34375  
## AUC: 0.6845238  
## Gini: 0.3690476  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 5 11 0.687500 =11/16  
## 211 0 21 0.000000 =0/21  
## Totals 5 32 0.297297 =11/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.187928 0.792453 10  
## 2 max f2 0.187928 0.905172 10  
## 3 max f0point5 0.187928 0.704698 10  
## 4 max accuracy 0.187928 0.702703 10  
## 5 max precision 0.998246 1.000000 0  
## 6 max recall 0.187928 1.000000 10  
## 7 max specificity 0.998246 1.000000 0  
## 8 max absolute\_mcc 0.187928 0.452856 10  
## 9 max min\_per\_class\_accuracy 0.627600 0.333333 9  
## 10 max mean\_per\_class\_accuracy 0.187928 0.656250 10  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_4)



## Rectifier with Dropout:

nn\_5 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_5",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "RectifierWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

##   
 |   
 | | 0%  
 |   
 |= | 1%  
 |   
 |== | 3%  
 |   
 |=== | 4%  
 |   
 |=== | 5%  
 |   
 |==== | 6%  
 |   
 |==== | 7%  
 |   
 |===== | 7%  
 |   
 |===== | 8%  
 |   
 |====== | 9%  
 |   
 |====== | 10%  
 |   
 |======= | 10%  
 |   
 |======= | 11%  
 |   
 |======== | 12%  
 |   
 |========= | 13%  
 |   
 |========= | 14%  
 |   
 |========== | 15%  
 |   
 |========== | 16%  
 |   
 |=========== | 16%  
 |   
 |=========== | 17%  
 |   
 |============ | 18%  
 |   
 |============ | 19%  
 |   
 |============= | 20%  
 |   
 |============== | 21%  
 |   
 |============== | 22%  
 |   
 |=============== | 23%  
 |   
 |=============== | 24%  
 |   
 |================ | 24%  
 |   
 |================ | 25%  
 |   
 |================= | 26%  
 |   
 |================== | 27%  
 |   
 |================== | 28%  
 |   
 |=================== | 29%  
 |   
 |=================== | 30%  
 |   
 |==================== | 30%  
 |   
 |==================== | 31%  
 |   
 |===================== | 32%  
 |   
 |===================== | 33%  
 |   
 |======================= | 35%  
 |   
 |======================= | 36%  
 |   
 |======================== | 37%  
 |   
 |========================= | 38%  
 |   
 |========================== | 39%  
 |   
 |========================== | 40%  
 |   
 |========================== | 41%  
 |   
 |=========================== | 42%  
 |   
 |============================ | 42%  
 |   
 |============================ | 43%  
 |   
 |============================ | 44%  
 |   
 |============================= | 44%  
 |   
 |============================= | 45%  
 |   
 |============================== | 46%  
 |   
 |=============================== | 47%  
 |   
 |=============================== | 48%  
 |   
 |================================ | 49%  
 |   
 |================================ | 50%  
 |   
 |================================= | 50%  
 |   
 |================================= | 51%  
 |   
 |================================== | 52%  
 |   
 |=================================== | 53%  
 |   
 |=================================== | 54%  
 |   
 |==================================== | 55%  
 |   
 |==================================== | 56%  
 |   
 |===================================== | 56%  
 |   
 |===================================== | 57%  
 |   
 |===================================== | 58%  
 |   
 |====================================== | 58%  
 |   
 |======================================= | 59%  
 |   
 |======================================= | 60%  
 |   
 |======================================= | 61%  
 |   
 |======================================== | 61%  
 |   
 |======================================== | 62%  
 |   
 |========================================= | 63%  
 |   
 |========================================== | 65%  
 |   
 |=========================================== | 66%  
 |   
 |=========================================== | 67%  
 |   
 |============================================ | 67%  
 |   
 |============================================ | 68%  
 |   
 |============================================= | 69%  
 |   
 |============================================= | 70%  
 |   
 |============================================== | 71%  
 |   
 |=============================================== | 72%  
 |   
 |================================================ | 73%  
 |   
 |================================================ | 74%  
 |   
 |================================================= | 75%  
 |   
 |================================================== | 76%  
 |   
 |================================================== | 77%  
 |   
 |=================================================== | 78%  
 |   
 |=================================================== | 79%  
 |   
 |==================================================== | 79%  
 |   
 |==================================================== | 80%  
 |   
 |==================================================== | 81%  
 |   
 |===================================================== | 81%  
 |   
 |===================================================== | 82%  
 |   
 |====================================================== | 83%  
 |   
 |======================================================= | 85%  
 |   
 |======================================================== | 87%  
 |   
 |========================================================= | 87%  
 |   
 |========================================================= | 88%  
 |   
 |========================================================== | 89%  
 |   
 |========================================================== | 90%  
 |   
 |=========================================================== | 91%  
 |   
 |============================================================ | 92%  
 |   
 |============================================================ | 93%  
 |   
 |============================================================= | 93%  
 |   
 |============================================================= | 94%  
 |   
 |============================================================== | 95%  
 |   
 |=============================================================== | 96%  
 |   
 |=============================================================== | 97%  
 |   
 |=============================================================== | 98%  
 |   
 |================================================================ | 98%  
 |   
 |================================================================ | 99%  
 |   
 |=================================================================| 99%  
 |   
 |=================================================================| 100%

h2o.performance(nn\_5,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.193127  
## RMSE: 0.4394622  
## LogLoss: 2.171004  
## Mean Per-Class Error: 0.3125  
## AUC: 0.7261905  
## Gini: 0.452381  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 6 10 0.625000 =10/16  
## 211 0 21 0.000000 =0/21  
## Totals 6 31 0.270270 =10/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.586526 0.807692 7  
## 2 max f2 0.586526 0.913043 7  
## 3 max f0point5 0.586526 0.724138 7  
## 4 max accuracy 0.586526 0.729730 7  
## 5 max precision 1.000000 0.833333 3  
## 6 max recall 0.586526 1.000000 7  
## 7 max specificity 1.000000 0.937500 0  
## 8 max absolute\_mcc 0.586526 0.504016 7  
## 9 max min\_per\_class\_accuracy 0.586526 0.375000 7  
## 10 max mean\_per\_class\_accuracy 0.586526 0.687500 7  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_5)



# Maxout with Dropout:

nn\_6 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_6",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "MaxoutWithDropout",  
 hidden = c(500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T  
)

##   
 |   
 | | 0%  
 |   
 |== | 3%  
 |   
 |=== | 4%  
 |   
 |=== | 5%  
 |   
 |==== | 6%  
 |   
 |==== | 7%  
 |   
 |===== | 7%  
 |   
 |===== | 8%  
 |   
 |====== | 10%  
 |   
 |======= | 10%  
 |   
 |======= | 11%  
 |   
 |======== | 13%  
 |   
 |========= | 14%  
 |   
 |========== | 15%  
 |   
 |========== | 16%  
 |   
 |=========== | 16%  
 |   
 |=========== | 17%  
 |   
 |=========== | 18%  
 |   
 |============ | 18%  
 |   
 |============ | 19%  
 |   
 |============= | 19%  
 |   
 |============= | 20%  
 |   
 |============== | 21%  
 |   
 |============== | 22%  
 |   
 |=============== | 22%  
 |   
 |=============== | 23%  
 |   
 |=============== | 24%  
 |   
 |================ | 24%  
 |   
 |================ | 25%  
 |   
 |================= | 25%  
 |   
 |================= | 26%  
 |   
 |================= | 27%  
 |   
 |================== | 28%  
 |   
 |=================== | 29%  
 |   
 |=================== | 30%  
 |   
 |==================== | 31%  
 |   
 |===================== | 32%  
 |   
 |===================== | 33%  
 |   
 |====================== | 34%  
 |   
 |======================= | 35%  
 |   
 |======================= | 36%  
 |   
 |======================== | 37%  
 |   
 |========================= | 39%  
 |   
 |========================== | 39%  
 |   
 |=========================== | 41%  
 |   
 |=========================== | 42%  
 |   
 |============================ | 43%  
 |   
 |============================ | 44%  
 |   
 |============================= | 45%  
 |   
 |============================== | 46%  
 |   
 |============================== | 47%  
 |   
 |=============================== | 48%  
 |   
 |================================ | 49%  
 |   
 |================================ | 50%  
 |   
 |================================= | 51%  
 |   
 |================================== | 52%  
 |   
 |================================== | 53%  
 |   
 |=================================== | 54%  
 |   
 |==================================== | 55%  
 |   
 |==================================== | 56%  
 |   
 |===================================== | 57%  
 |   
 |====================================== | 58%  
 |   
 |====================================== | 59%  
 |   
 |======================================= | 60%  
 |   
 |======================================== | 61%  
 |   
 |======================================== | 62%  
 |   
 |========================================= | 63%  
 |   
 |========================================== | 64%  
 |   
 |========================================== | 65%  
 |   
 |=========================================== | 66%  
 |   
 |============================================ | 67%  
 |   
 |============================================ | 68%  
 |   
 |============================================= | 69%  
 |   
 |============================================== | 70%  
 |   
 |============================================== | 71%  
 |   
 |=============================================== | 72%  
 |   
 |=============================================== | 73%  
 |   
 |================================================ | 74%  
 |   
 |================================================= | 75%  
 |   
 |================================================= | 76%  
 |   
 |================================================== | 76%  
 |   
 |================================================== | 77%  
 |   
 |================================================== | 78%  
 |   
 |=================================================== | 79%  
 |   
 |==================================================== | 79%  
 |   
 |==================================================== | 80%  
 |   
 |==================================================== | 81%  
 |   
 |===================================================== | 81%  
 |   
 |===================================================== | 82%  
 |   
 |====================================================== | 82%  
 |   
 |====================================================== | 83%  
 |   
 |====================================================== | 84%  
 |   
 |======================================================= | 84%  
 |   
 |======================================================= | 85%  
 |   
 |======================================================== | 86%  
 |   
 |======================================================== | 87%  
 |   
 |========================================================= | 88%  
 |   
 |========================================================== | 89%  
 |   
 |========================================================== | 90%  
 |   
 |=========================================================== | 90%  
 |   
 |=========================================================== | 91%  
 |   
 |============================================================ | 92%  
 |   
 |============================================================ | 93%  
 |   
 |============================================================= | 93%  
 |   
 |============================================================= | 94%  
 |   
 |============================================================== | 95%  
 |   
 |=============================================================== | 96%  
 |   
 |=============================================================== | 97%  
 |   
 |================================================================ | 98%  
 |   
 |=================================================================| 99%  
 |   
 |=================================================================| 100%

h2o.performance(nn\_6,test.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.4449376  
## RMSE: 0.6670364  
## LogLoss: 2.888669  
## Mean Per-Class Error: 0.3988095  
## AUC: 0.610119  
## Gini: 0.2202381  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 4 12 0.750000 =12/16  
## 211 1 20 0.047619 =1/21  
## Totals 5 32 0.351351 =13/37  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.019517 0.754717 10  
## 2 max f2 0.000000 0.867769 15  
## 3 max f0point5 0.019517 0.671141 10  
## 4 max accuracy 0.019517 0.648649 10  
## 5 max precision 1.000000 1.000000 0  
## 6 max recall 0.000000 1.000000 15  
## 7 max specificity 1.000000 1.000000 0  
## 8 max absolute\_mcc 0.999996 0.303895 3  
## 9 max min\_per\_class\_accuracy 0.998602 0.285714 7  
## 10 max mean\_per\_class\_accuracy 0.019517 0.601190 10  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_6)



# h2o.saveModel(nn\_1,"nn1.RData")  
# h2o.saveModel(nn\_2,"nn2.RData")  
# h2o.saveModel(nn\_3,"nn3.RData")  
# h2o.saveModel(nn\_4,"nn4.RData")  
# h2o.saveModel(nn\_5,"nn5.RData")  
# h2o.saveModel(nn\_6,"nn6.RData")

h2o.performance(nn\_1,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1625316  
## RMSE: 0.4031521  
## LogLoss: 0.8764427  
## Mean Per-Class Error: 0.2624625  
## AUC: 0.8447447  
## Gini: 0.6894895  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 46 44 0.488889 =44/90  
## 211 4 107 0.036036 =4/111  
## Totals 50 151 0.238806 =48/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.023067 0.816794 44  
## 2 max f2 0.000234 0.908333 49  
## 3 max f0point5 0.542199 0.795107 41  
## 4 max accuracy 0.416495 0.761194 42  
## 5 max precision 0.542199 0.962963 41  
## 6 max recall 0.000000 1.000000 72  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.000234 0.548460 49  
## 9 max min\_per\_class\_accuracy 0.416495 0.522222 42  
## 10 max mean\_per\_class\_accuracy 0.416495 0.738589 42  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_2,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2387207  
## RMSE: 0.4885905  
## LogLoss: 0.9372256  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8333333  
## Gini: 0.6666667  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.120827 0.817844 59  
## 2 max f2 0.120827 0.913621 59  
## 3 max f0point5 0.862533 0.796460 52  
## 4 max accuracy 0.120827 0.756219 59  
## 5 max precision 0.985880 0.959184 44  
## 6 max recall 0.000173 1.000000 85  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.120827 0.554959 59  
## 9 max min\_per\_class\_accuracy 0.551938 0.504505 56  
## 10 max mean\_per\_class\_accuracy 0.120827 0.728829 59  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_3,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1475263  
## RMSE: 0.3840915  
## LogLoss: 0.658837  
## Mean Per-Class Error: 0.2701201  
## AUC: 0.8379379  
## Gini: 0.6758759  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 2 109 0.018018 =2/111  
## Totals 45 156 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.566689 0.816479 61  
## 2 max f2 0.566689 0.908333 61  
## 3 max f0point5 0.954867 0.805970 55  
## 4 max accuracy 0.566689 0.756219 61  
## 5 max precision 0.999615 0.981132 52  
## 6 max recall 0.000000 1.000000 89  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.566689 0.548460 61  
## 9 max min\_per\_class\_accuracy 0.903940 0.495495 57  
## 10 max mean\_per\_class\_accuracy 0.954867 0.732132 55  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_4,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2196784  
## RMSE: 0.4686986  
## LogLoss: 0.6327696  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8318318  
## Gini: 0.6636637  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.155375 0.817844 63  
## 2 max f2 0.155375 0.913621 63  
## 3 max f0point5 0.746435 0.785498 54  
## 4 max accuracy 0.155375 0.756219 63  
## 5 max precision 0.998246 1.000000 0  
## 6 max recall 0.009154 1.000000 84  
## 7 max specificity 0.998246 1.000000 0  
## 8 max absolute\_mcc 0.155375 0.554959 63  
## 9 max min\_per\_class\_accuracy 0.226987 0.495495 61  
## 10 max mean\_per\_class\_accuracy 0.155375 0.728829 63  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_5,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1436784  
## RMSE: 0.3790493  
## LogLoss: 0.7531213  
## Mean Per-Class Error: 0.2656156  
## AUC: 0.8436436  
## Gini: 0.6872873  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 1 110 0.009009 =1/111  
## Totals 44 157 0.238806 =48/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.084465 0.820896 46  
## 2 max f2 0.084465 0.915141 46  
## 3 max f0point5 0.956244 0.801749 41  
## 4 max accuracy 0.586526 0.761194 44  
## 5 max precision 1.000000 0.960000 8  
## 6 max recall 0.005610 1.000000 52  
## 7 max specificity 1.000000 0.988889 0  
## 8 max absolute\_mcc 0.084465 0.563722 46  
## 9 max min\_per\_class\_accuracy 0.956244 0.495495 41  
## 10 max mean\_per\_class\_accuracy 0.586526 0.735435 44  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

h2o.performance(nn\_6,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.2847582  
## RMSE: 0.5336274  
## LogLoss: 1.411247  
## Mean Per-Class Error: 0.2746246  
## AUC: 0.832032  
## Gini: 0.6640641  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 43 47 0.522222 =47/90  
## 211 3 108 0.027027 =3/111  
## Totals 46 155 0.248756 =50/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.019517 0.812030 59  
## 2 max f2 0.019517 0.901503 59  
## 3 max f0point5 0.514351 0.805970 54  
## 4 max accuracy 0.019517 0.751244 59  
## 5 max precision 1.000000 1.000000 0  
## 6 max recall 0.000000 1.000000 104  
## 7 max specificity 1.000000 1.000000 0  
## 8 max absolute\_mcc 0.019517 0.533549 59  
## 9 max min\_per\_class\_accuracy 0.514351 0.486486 54  
## 10 max mean\_per\_class\_accuracy 0.514351 0.732132 54  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

# Recitfier with Dropout with 2 additional hidden layers of 500 neurons each:

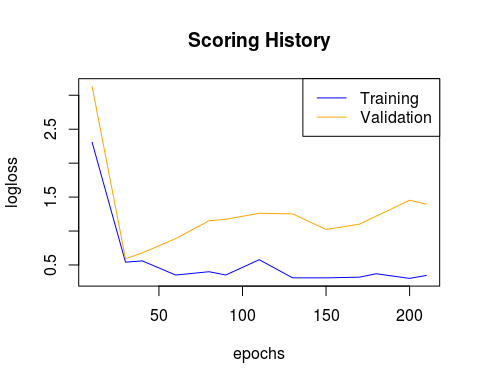
nn\_7 <- h2o.deeplearning(  
 x= 2:97,  
 y= 1,  
 model\_id = "nn\_7",  
 training\_frame = train.hex,  
 validation\_frame = valid.hex,  
 activation = "RectifierWithDropout",  
 hidden = c(500,500,500,500),  
 nfolds=10,  
 standardize = F,  
 fast\_mode = F,  
 sparse = T,  
 epochs=200,  
 balance\_classes = T,  
 l1=1e-06  
)

##   
 |   
 | | 0%  
 |   
 |= | 1%  
 |   
 |== | 3%  
 |   
 |== | 4%  
 |   
 |=== | 4%  
 |   
 |=== | 5%  
 |   
 |==== | 6%  
 |   
 |==== | 7%  
 |   
 |===== | 7%  
 |   
 |===== | 8%  
 |   
 |====== | 9%  
 |   
 |====== | 10%  
 |   
 |======= | 10%  
 |   
 |======= | 11%  
 |   
 |======== | 12%  
 |   
 |======== | 13%  
 |   
 |========= | 14%  
 |   
 |========= | 15%  
 |   
 |========== | 15%  
 |   
 |========== | 16%  
 |   
 |=========== | 17%  
 |   
 |============ | 18%  
 |   
 |============ | 19%  
 |   
 |============= | 19%  
 |   
 |============= | 20%  
 |   
 |============== | 21%  
 |   
 |============== | 22%  
 |   
 |=============== | 23%  
 |   
 |================ | 24%  
 |   
 |================ | 25%  
 |   
 |================= | 26%  
 |   
 |================= | 27%  
 |   
 |================== | 27%  
 |   
 |================== | 28%  
 |   
 |=================== | 29%  
 |   
 |=================== | 30%  
 |   
 |==================== | 30%  
 |   
 |==================== | 31%  
 |   
 |===================== | 32%  
 |   
 |===================== | 33%  
 |   
 |====================== | 33%  
 |   
 |====================== | 34%  
 |   
 |====================== | 35%  
 |   
 |======================= | 35%  
 |   
 |======================= | 36%  
 |   
 |======================== | 36%  
 |   
 |======================== | 37%  
 |   
 |======================== | 38%  
 |   
 |========================= | 38%  
 |   
 |========================= | 39%  
 |   
 |========================== | 39%  
 |   
 |========================== | 40%  
 |   
 |========================== | 41%  
 |   
 |=========================== | 41%  
 |   
 |=========================== | 42%  
 |   
 |============================ | 43%  
 |   
 |============================= | 44%  
 |   
 |============================= | 45%  
 |   
 |============================== | 46%  
 |   
 |============================== | 47%  
 |   
 |=============================== | 47%  
 |   
 |=============================== | 48%  
 |   
 |================================ | 49%  
 |   
 |================================ | 50%  
 |   
 |================================= | 50%  
 |   
 |================================= | 51%  
 |   
 |================================== | 52%  
 |   
 |================================== | 53%  
 |   
 |=================================== | 53%  
 |   
 |=================================== | 54%  
 |   
 |==================================== | 55%  
 |   
 |==================================== | 56%  
 |   
 |===================================== | 56%  
 |   
 |===================================== | 57%  
 |   
 |===================================== | 58%  
 |   
 |====================================== | 58%  
 |   
 |====================================== | 59%  
 |   
 |======================================= | 59%  
 |   
 |======================================= | 60%  
 |   
 |======================================= | 61%  
 |   
 |======================================== | 61%  
 |   
 |======================================== | 62%  
 |   
 |========================================= | 62%  
 |   
 |========================================= | 63%  
 |   
 |========================================= | 64%  
 |   
 |========================================== | 64%  
 |   
 |========================================== | 65%  
 |   
 |=========================================== | 66%  
 |   
 |=========================================== | 67%  
 |   
 |============================================ | 67%  
 |   
 |============================================ | 68%  
 |   
 |============================================= | 69%  
 |   
 |============================================= | 70%  
 |   
 |============================================== | 70%  
 |   
 |============================================== | 71%  
 |   
 |=============================================== | 72%  
 |   
 |=============================================== | 73%  
 |   
 |================================================ | 74%  
 |   
 |================================================= | 75%  
 |   
 |================================================= | 76%  
 |   
 |================================================== | 77%  
 |   
 |=================================================== | 78%  
 |   
 |=================================================== | 79%  
 |   
 |==================================================== | 80%  
 |   
 |===================================================== | 81%  
 |   
 |===================================================== | 82%  
 |   
 |====================================================== | 83%  
 |   
 |======================================================= | 84%  
 |   
 |======================================================= | 85%  
 |   
 |======================================================== | 86%  
 |   
 |========================================================= | 87%  
 |   
 |========================================================= | 88%  
 |   
 |========================================================== | 89%  
 |   
 |========================================================== | 90%  
 |   
 |=========================================================== | 90%  
 |   
 |=========================================================== | 91%  
 |   
 |=========================================================== | 92%  
 |   
 |============================================================ | 92%  
 |   
 |============================================================ | 93%  
 |   
 |============================================================= | 93%  
 |   
 |============================================================= | 94%  
 |   
 |============================================================== | 95%  
 |   
 |============================================================== | 96%  
 |   
 |=============================================================== | 96%  
 |   
 |=============================================================== | 97%  
 |   
 |=============================================================== | 98%  
 |   
 |================================================================ | 98%  
 |   
 |================================================================ | 99%  
 |   
 |=================================================================| 99%  
 |   
 |=================================================================| 100%

# h2o.saveModel(nn\_7,"nn7.RData")  
  
h2o.performance(nn\_7,dat.hex)

## H2OBinomialMetrics: deeplearning  
##   
## MSE: 0.1582569  
## RMSE: 0.3978151  
## LogLoss: 0.9577213  
## Mean Per-Class Error: 0.2711712  
## AUC: 0.8334334  
## Gini: 0.6668669  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## -211 211 Error Rate  
## -211 42 48 0.533333 =48/90  
## 211 1 110 0.009009 =1/111  
## Totals 43 158 0.243781 =49/201  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.695753 0.817844 51  
## 2 max f2 0.695753 0.913621 51  
## 3 max f0point5 1.000000 0.796460 44  
## 4 max accuracy 0.695753 0.756219 51  
## 5 max precision 1.000000 0.951220 28  
## 6 max recall 0.000113 1.000000 60  
## 7 max specificity 1.000000 0.977778 0  
## 8 max absolute\_mcc 0.695753 0.554959 51  
## 9 max min\_per\_class\_accuracy 0.934974 0.504505 48  
## 10 max mean\_per\_class\_accuracy 0.695753 0.728829 51  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

plot(nn\_7)



# Ensemble:

Add the predictions from all 7 neural networks as features to dataset:

# nn\_1 <- h2o.loadModel("nn\_1")  
# nn\_2 <- h2o.loadModel("nn\_2")  
# nn\_3 <- h2o.loadModel("nn\_3")  
# nn\_4 <- h2o.loadModel("nn\_4")  
# nn\_5 <- h2o.loadModel("nn\_5")  
# nn\_6 <- h2o.loadModel("nn\_6")  
# nn\_7 <- h2o.loadModel("nn\_7")  
#   
p1 <- as.data.frame(h2o.predict(nn\_1,dat.hex))

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

p2 <- as.data.frame(h2o.predict(nn\_2,dat.hex))

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

p3 <- as.data.frame(h2o.predict(nn\_3,dat.hex))

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

p4 <- as.data.frame(h2o.predict(nn\_4,dat.hex))

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

p5 <- as.data.frame(h2o.predict(nn\_5,dat.hex))

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

p6 <- as.data.frame(h2o.predict(nn\_6,dat.hex))

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

p7 <- as.data.frame(h2o.predict(nn\_7,dat.hex))

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

p1 <- p1[,3]  
p2 <- p2[,3]  
p3 <- p3[,3]  
p4 <- p4[,3]  
p5 <- p5[,3]  
p6 <- p6[,3]  
p7 <- p7[,3]  
  
p <- data.frame(cbind(p1,p2,p3,p4,p5,p6,p7))  
  
p <- scale(p)  
  
my.dat <- data.frame(cbind(my.dat,p))

h2o.shutdown(prompt=F)

## [1] TRUE

# h2o.init(max\_mem\_size = "28G",nthreads = -1)  
# h2o.removeAll()  
# #h2o.no\_progress()  
#   
# dat.hex <- as.h2o(my.dat,"dat.hex")  
#   
# splitz <- h2o.splitFrame(dat.hex,ratios=c(0.6,0.2),  
# destination\_frames = c("train.hex","valid.hex","test.hex"))  
#   
# train.hex <- splitz[[1]]  
# valid.hex <- splitz[[2]]  
# test.hex <- splitz[[3]]

## Random Forests:

Predict using randomforests, and add its prediction as a feature:

# rf1 <- h2o.randomForest(y=1,  
# x=2:111,  
# training\_frame=train.hex,  
# validation\_frame=valid.hex,  
# nfolds=30,  
# ntrees=200)  
#   
# h2o.performance(rf1,dat.hex)  
#   
# p8 <- as.data.frame(h2o.predict(rf1,dat.hex))  
#   
# plot(rf1)  
#   
# p8 <- p8[,3]  
# my.dat <- data.frame(my.dat,p8)  
# #   
# h2o.shutdown(prompt = F)

## K-means clustering

Perform k-means clustering, with k=10, and add cluster group as a variable:

rm(list=ls())  
load("mydat.RData")  
k <- kmeans(x=my.dat[,-1],centers=10)  
k <- k$cluster  
k <- as.factor(k)  
  
require(dummies)  
  
k <- dummy(k)  
  
my.dat <- data.frame(cbind(my.dat,k))  
  
my.dat$k1 <- as.factor(my.dat$k1)  
my.dat$k2 <- as.factor(my.dat$k2)  
my.dat$k3 <- as.factor(my.dat$k3)  
my.dat$k4 <- as.factor(my.dat$k4)  
my.dat$k5 <- as.factor(my.dat$k5)  
my.dat$k6 <- as.factor(my.dat$k6)  
my.dat$k7 <- as.factor(my.dat$k7)  
my.dat$k8 <- as.factor(my.dat$k8)  
my.dat$k9 <- as.factor(my.dat$k9)  
my.dat$k10 <- as.factor(my.dat$k10)  
save(my.dat,file="mydat.RData")

rm(list=ls())  
load("mydat.RData")

## Support Vector Machines

Predict using various kernels and add as features:

Linear Kernel:

require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="linear")  
svm.p <- data.frame(predict(sv,my.dat))

Polynomial Kernel:

sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="polynomial")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

Radial Kernel

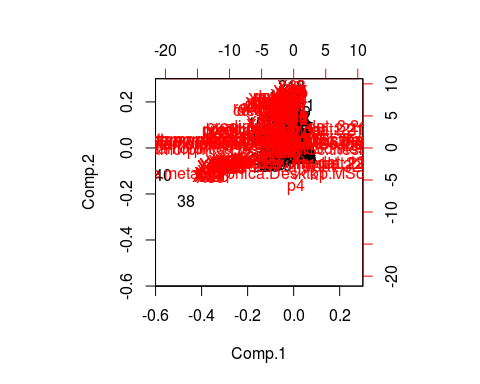
require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="radial")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

Sigmoid Kernel:

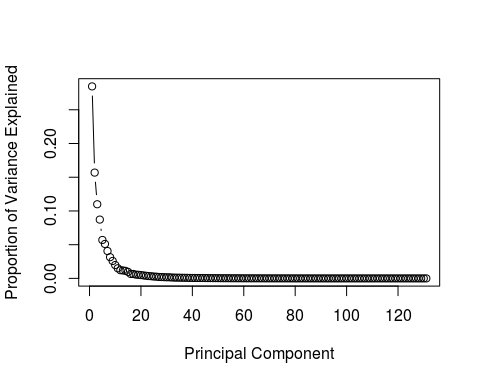
require(e1071)  
sv <- svm(pdgCode~.,data=my.dat,scale=F,kernel="sigmoid")  
svm.p <- data.frame(cbind(svm.p,predict(sv,my.dat)))

require(dummies)  
svm.p <- as.data.frame(svm.p)  
svm.p <- dummy.data.frame(svm.p)  
my.dat <- data.frame(cbind(my.dat,svm.p))

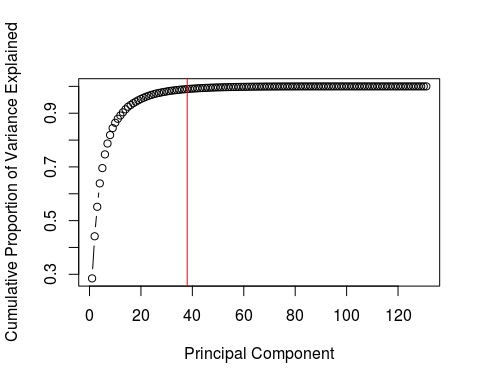
my.dat$k1 <- as.numeric(my.dat$k1)  
my.dat$k2 <- as.numeric(my.dat$k2)  
my.dat$k3 <- as.numeric(my.dat$k3)  
my.dat$k4 <- as.numeric(my.dat$k4)  
my.dat$k5 <- as.numeric(my.dat$k5)  
my.dat$k6 <- as.numeric(my.dat$k6)  
my.dat$k7 <- as.numeric(my.dat$k7)  
my.dat$k8 <- as.numeric(my.dat$k8)  
my.dat$k9 <- as.numeric(my.dat$k9)  
my.dat$k10 <- as.numeric(my.dat$k10)  
  
pc <- princomp(my.dat[,-1])  
biplot(pc)



std\_dev <- pc$sdev  
  
pr\_var <- std\_dev^2  
  
prop\_varex <- pr\_var/sum(pr\_var)  
  
plot(prop\_varex, xlab = "Principal Component",  
 ylab = "Proportion of Variance Explained",  
 type = "b")



plot(cumsum(prop\_varex), xlab = "Principal Component",  
 ylab = "Cumulative Proportion of Variance Explained",  
 type = "b")  
  
abline(v=min(which(cumsum(prop\_varex)>=.99)),col="red")



min(which(cumsum(prop\_varex)>=.99))

## [1] 38

new.dat <- data.frame(cbind(as.character(my.dat$pdgCode),pc$scores[,1:38]))  
new.dat$V1 <- as.factor(new.dat$V1)  
names(new.dat)[1] <- "pdgCode"  
  
for(i in 2:ncol(new.dat)){  
 new.dat[,i] <- as.numeric(as.character(new.dat[,i]))  
}

# Final prediction

Build a random forest that uses all the new features, as well as the original dataset to classify particles as pdgcode 211 or pdgcode -211:

require(randomForest)  
  
rf <- randomForest(x=new.dat[1:100,2:39],y=new.dat$pdgCode[1:100])  
  
pp <- predict(rf,newdata = new.dat)  
  
pp <- data.frame(cbind(as.factor(new.dat$pdgCode),as.factor(pp)))  
pp[,3] <- pp[,1]==pp[,2]  
  
length(which(pp[,3])==T)/nrow(pp)

## [1] 0.7562189