

Neural Networks

Shantam Gupta

May 13, 2018

Contents

1	Installing the Package	1
2	Load the Data	1
3	Preprocess the data: Normalize the data	1
4	Building Neural Network	2
5	build the mlp(multi layer perceptron) deep learning model2 using h2o	6
6	build the mlp(multi layer perceptron) deep learning model using h2o	12
6.1	Main Data	16
7	Simulation : Spike in mean X 10 per feature to check variable importance	17
8	Model with 12 Effects	37

1 Installing the Package

2 Load the Data

3 Preprocess the data: Normalize the data

```
new_data <- rbind(S0,Data)
maxs <- apply(new_data %>% dplyr::select(-c(idfile,RESPONSE)), 2, max)
mins <- apply(new_data %>% dplyr::select(-c(idfile,RESPONSE)), 2, min)

scaled_data <- as.data.frame(scale(new_data %>% dplyr::select(-c(idfile,RESPONSE)), center = mins, scale = maxs)
#scaled_data$RESPONSE <- ifelse(new_data$RESPONSE == "GO", 1, 0)
scaled_data$RESPONSE <- as.factor(new_data$RESPONSE)
#scaled_data$RESPONSE <- as.factor(scaled_data$RESPONSE)
scaled_data$idfile <- new_data$idfile

#select random ind for train and test
set.seed(123)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(scaled_data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(scaled_data)), size = smp_size)
```

```
train <- scaled_data[train_ind,]
test <- scaled_data[-train_ind,]
```

4 Building Neural Network

```
library(h2o)

##
## -----
##
## Your next step is to start H2O:
##   > h2o.init()
##
## For H2O package documentation, ask for help:
##   > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## -----
##
## Attaching package: 'h2o'
##
## The following objects are masked from 'package:stats':
##
##   cor, sd, var
##
## The following objects are masked from 'package:base':
##
##   %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##   colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##   log10, log1p, log2, round, signif, trunc
##
#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

#import r objects to h2o cloud
train_h2o <- as.h2o(train)
test_h2o <- as.h2o(test)

#build the mlp(multi layer perceptron) deep learning model using h2o
set.seed(100)

dl_model <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
```

```

x= colnames(train_h2o[,1:48]),
y= "RESPONSE",
activation="Tanh",
hidden=c(5,4),
stopping_metric="mean_per_class_error",
stopping_tolerance=0.01,
epochs=100,
seed = 123, # give seed
export_weights_and_biases = T, # export weights and biases defaults to false
reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)

```

```
summary(dl_model)
```

```

## Model Details:
## =====
##
## H2OBinomialModel: deeplearning
## Model Key: dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossE
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1    48   Input  0.00 %
## 2      2     5   Tanh  0.00 % 0.000000 0.000000  0.002543 0.002428
## 3      3     4   Tanh  0.00 % 0.000000 0.000000  0.003763 0.004709
## 4      4     2 Softmax      0.000000 0.000000  0.002780 0.000511
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000  -0.008720   0.196638 -0.038391 0.123508
## 3 0.000000  -0.032180   0.416094  0.235844 0.537879
## 4 0.000000  -0.194668   3.183959  0.000000 0.637211
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.02160276
## RMSE:  0.1469788
## LogLoss:  0.07299448
## Mean Per-Class Error:  0.1421513
## AUC:  0.9892086
## Gini:  0.9784173
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO   Error   Rate
## GO      60   23 0.277108   =23/83
## NOGO     7  966 0.007194   =7/973
## Totals 67  989 0.028409  =30/1056
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##
##      metric threshold   value idx
## 1      max f1  0.336047 0.984709 332
## 2      max f2  0.336047 0.989551 332
## 3      max f0point5 0.800684 0.988976 280

```

```

## 4          max accuracy 0.458565 0.971591 319
## 5          max precision 0.999996 1.000000 0
## 6          max recall 0.088233 1.000000 386
## 7          max specificity 0.999996 1.000000 0
## 8          max absolute_mcc 0.641986 0.808694 303
## 9  max min_per_class_accuracy 0.800684 0.958890 280
## 10 max mean_per_class_accuracy 0.800684 0.961373 280
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/I>)`
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE: 0.02120419
## RMSE: 0.1456166
## LogLoss: 0.07595011
## Mean Per-Class Error: 0.06857903
## AUC: 0.9840426
## Gini: 0.9680851
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      21      3 0.125000  =3/24
## NOGO      4    325 0.012158  =4/329
## Totals 25    328 0.019830  =7/353
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##
##      metric threshold      value idx
## 1          max f1 0.449576 0.989346 326
## 2          max f2 0.281052 0.989709 334
## 3          max f0point5 0.449576 0.990250 326
## 4          max accuracy 0.449576 0.980170 326
## 5          max precision 0.999999 1.000000 0
## 6          max recall 0.051224 1.000000 350
## 7          max specificity 0.999999 1.000000 0
## 8          max absolute_mcc 0.449576 0.846697 326
## 9  max min_per_class_accuracy 0.925779 0.916667 302
## 10 max mean_per_class_accuracy 0.966708 0.946809 292
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/I>)`
##
##
## Scoring History:
##      timestamp      duration training_speed      epochs iterations
## 1 2018-06-05 11:22:14 0.000 sec          0.000000          0
## 2 2018-06-05 11:22:14 0.047 sec    62117 obs/sec    1.000000          1
## 3 2018-06-05 11:22:15 0.085 sec    54153 obs/sec    2.000000          2
## 4 2018-06-05 11:22:15 0.122 sec    48000 obs/sec    3.000000          3
## 5 2018-06-05 11:22:15 0.153 sec    52148 obs/sec    4.000000          4
## 6 2018-06-05 11:22:15 0.200 sec    46725 obs/sec    5.000000          5
## 7 2018-06-05 11:22:15 0.231 sec    49116 obs/sec    6.000000          6
## 8 2018-06-05 11:22:15 0.285 sec    48631 obs/sec    7.000000          7
## 9 2018-06-05 11:22:15 0.321 sec    50586 obs/sec    8.000000          8
## 10 2018-06-05 11:22:15 0.354 sec    51934 obs/sec    9.000000          9

```

```

## 11 2018-06-05 11:22:15 0.400 sec 49345 obs/sec 10.00000 10
## 12 2018-06-05 11:22:15 0.432 sec 49220 obs/sec 11.00000 11
## 13 2018-06-05 11:22:15 0.469 sec 49307 obs/sec 12.00000 12
## 14 2018-06-05 11:22:15 0.500 sec 47666 obs/sec 13.00000 13
## 15 2018-06-05 11:22:15 0.547 sec 46200 obs/sec 14.00000 14
## 16 2018-06-05 11:22:15 0.585 sec 46315 obs/sec 15.00000 15
## 17 2018-06-05 11:22:15 0.619 sec 47195 obs/sec 16.00000 16
## 18 2018-06-05 11:22:15 0.654 sec 48650 obs/sec 17.00000 17
## 19 2018-06-05 11:22:15 0.685 sec 49500 obs/sec 18.00000 18
## 20 2018-06-05 11:22:15 0.724 sec 50285 obs/sec 19.00000 19
##      samples training_rmse training_logloss training_auc training_lift
## 1      0.000000
## 2    1056.000000      0.34876      0.41973      0.74189      1.08530
## 3    2112.000000      0.26678      0.22887      0.86701      1.08530
## 4    3168.000000      0.23017      0.17621      0.91605      1.08530
## 5    4224.000000      0.21174      0.15128      0.93714      1.08530
## 6    5280.000000      0.20034      0.13640      0.95234      1.08530
## 7    6336.000000      0.19395      0.12663      0.95937      1.08530
## 8    7392.000000      0.18517      0.11657      0.96674      1.08530
## 9    8448.000000      0.18058      0.11007      0.97101      1.08530
## 10   9504.000000      0.17723      0.10600      0.97429      1.08530
## 11  10560.000000      0.17258      0.09953      0.97789      1.08530
## 12  11616.000000      0.16842      0.09492      0.98023      1.08530
## 13  12672.000000      0.16619      0.09167      0.98191      1.08530
## 14  13728.000000      0.16203      0.08787      0.98331      1.08530
## 15  14784.000000      0.15879      0.08458      0.98484      1.08530
## 16  15840.000000      0.15608      0.08191      0.98608      1.08530
## 17  16896.000000      0.15388      0.07925      0.98705      1.08530
## 18  17952.000000      0.15095      0.07675      0.98803      1.08530
## 19  19008.000000      0.14907      0.07468      0.98873      1.08530
## 20  20064.000000      0.14698      0.07299      0.98921      1.08530
##      training_classification_error validation_rmse validation_logloss
## 1
## 2      0.07860      0.35161      0.41492
## 3      0.07670      0.26557      0.21827
## 4      0.06913      0.22815      0.16376
## 5      0.05871      0.21085      0.14001
## 6      0.05019      0.19750      0.12624
## 7      0.04640      0.19071      0.11648
## 8      0.04072      0.17763      0.10623
## 9      0.03883      0.17297      0.10066
## 10     0.03788      0.16468      0.09508
## 11     0.03598      0.16398      0.09187
## 12     0.03409      0.15909      0.08814
## 13     0.03409      0.15822      0.08670
## 14     0.03409      0.15207      0.08274
## 15     0.03220      0.15053      0.08088
## 16     0.03125      0.14833      0.07885
## 17     0.03030      0.14991      0.07936
## 18     0.02936      0.14661      0.07686
## 19     0.02936      0.14730      0.07686
## 20     0.02841      0.14562      0.07595
##      validation_auc validation_lift validation_classification_error
## 1

```

```
## 2      0.73252      1.07295      0.06516
## 3      0.86955      1.07295      0.06799
## 4      0.92705      1.07295      0.06516
## 5      0.94871      1.07295      0.05949
## 6      0.96137      1.07295      0.04533
## 7      0.96682      1.07295      0.03966
## 8      0.97264      1.07295      0.02833
## 9      0.97619      1.07295      0.02550
## 10     0.97809      1.07295      0.02550
## 11     0.97910      1.07295      0.02550
## 12     0.98050      1.07295      0.02266
## 13     0.98100      1.07295      0.02266
## 14     0.98265      1.07295      0.02266
## 15     0.98341      1.07295      0.01983
## 16     0.98417      1.07295      0.01983
## 17     0.98366      1.07295      0.02266
## 18     0.98379      1.07295      0.02266
## 19     0.98417      1.07295      0.02266
## 20     0.98404      1.07295      0.01983
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
##      variable relative_importance scaled_importance percentage
## 1  MassAccu.SLHTLFGDELCK      1.000000      1.000000  0.034675
## 2 MassAccu.ECCHGDLLECADDR      0.925454      0.925454  0.032090
## 3      RT.HLVDEPQNLIK      0.890437      0.890437  0.030876
## 4   Charge.VPQVSTPTLVEVSR      0.872659      0.872659  0.030259
## 5 TotalArea.SLHTLFGDELCK      0.860369      0.860369  0.029833
##
## ---
##      variable relative_importance scaled_importance percentage
## 43      RT.NECFLSHK      0.384706      0.384706  0.013340
## 44 MassAccu.EACFAVEGPK      0.356352      0.356352  0.012356
## 45      FWHM.EACFAVEGPK      0.344777      0.344777  0.011955
## 46      MZ.HLVDEPQNLIK      0.331787      0.331787  0.011505
## 47 Charge.SLHTLFGDELCK      0.322294      0.322294  0.011175
## 48      MZ.VPQVSTPTLVEVSR      0.267940      0.267940  0.009291
```

The accuracy is 98.57% . The net could be optimized further to improve the accuracy

5 build the mlp(multi layer perceptron) deep learning model2 using h2o

```
set.seed(100)

dl_model2 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:48]),
```

```

y= "RESPONSE",
activation="Tanh",
hidden=c(5,4),
stopping_metric="mean_per_class_error",
stopping_tolerance=0.01,
epochs=100,
seed = 123, # give seed
export_weights_and_biases = T, # export weights and biases defaults to false
reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)

```

5.0.1 Tuning the ANN

The simplest hyperparameter search method is a brute-force scan of the full Cartesian product of all combinations specified by a grid search. There are a lot of parameters to tune and due to limited computational capabilities we shall try to tune only some of them.

```

#hyperparameters to tune
hyper_params <- list(
  hidden=list(c(32,32,32),c(50,200,50)), # different architectures of hidden layer
  input_dropout_ratio=c(0,0.05), # values for drop out
  rate=c(0.01,0.02), # the learning rate
  activation = c("Rectifier") # activation functions
)

```

#grid search

```

grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id="dl_grid",
  model_id="dl_model_first",
  training_frame=train_h2o,
  x= colnames(train_h2o[,1:12]),
  y= "label",
  stopping_metric="mean_per_class_error",
  hyper_params = hyper_params,
  epochs=1000,
  stopping_tolerance=0.01,
  variable_importances=T
)

```

sort the model in the grid in decreasing order of error

```

grid <- h2o.getGrid("dl_grid", sort_by = "err", decreasing = FALSE)
grid

```

best model and its full set of parameters

```

grid@summary_table[1, ]
best_dl_model <- h2o.getModel(grid@model_ids[[1]])
best_dl_model

```

```

print(h2o.performance(best_dl_model))

```

storing the confusion matrix

```

best_dl_confusion <- as.data.frame(h2o.confusionMatrix(best_dl_model))

```

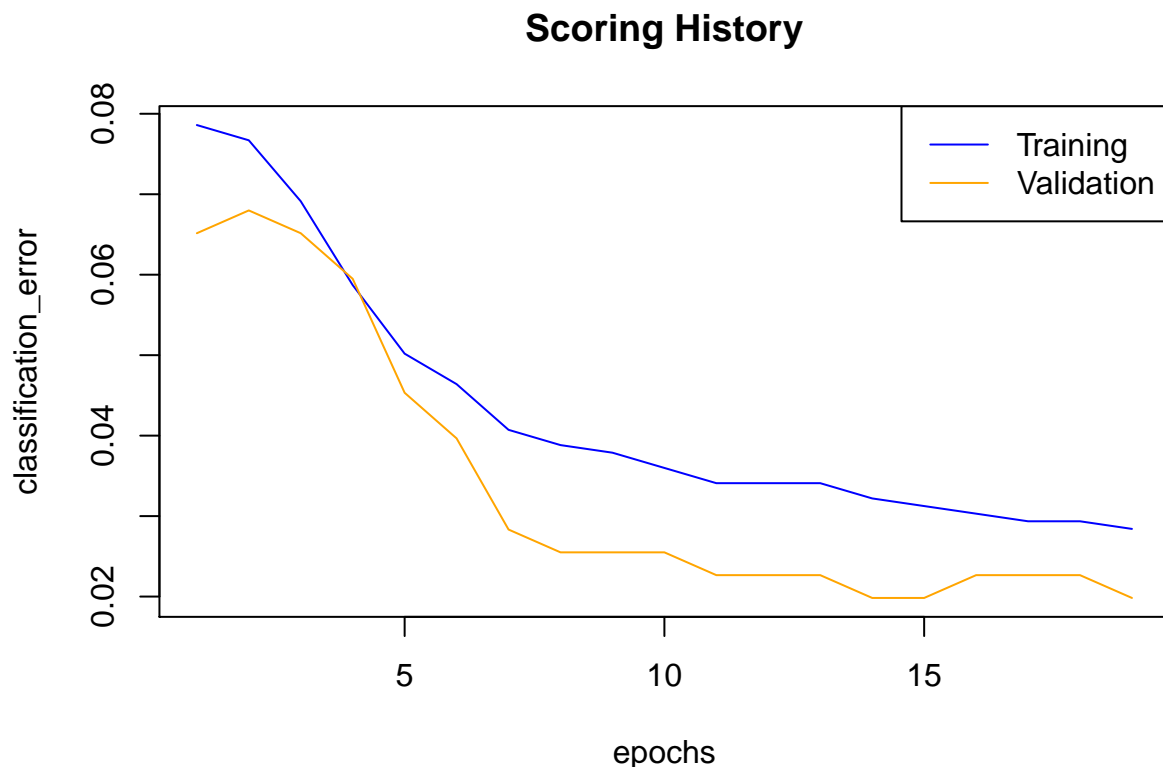
5.0.2 Plotting the model

```
plot(dl_model,timesteps = "epochs",metric = "classification_error")

## Warning in plot.window(...): "timesteps" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "timesteps" is not a graphical parameter
## Warning in title(...): "timesteps" is not a graphical parameter
## Warning in plot.window(...): "timesteps" is not a graphical parameter
## Warning in plot.xy(xy, type, ...): "timesteps" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...): "timesteps" is
## not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "timesteps" is
## not a graphical parameter

## Warning in box(...): "timesteps" is not a graphical parameter
## Warning in title(...): "timesteps" is not a graphical parameter
```



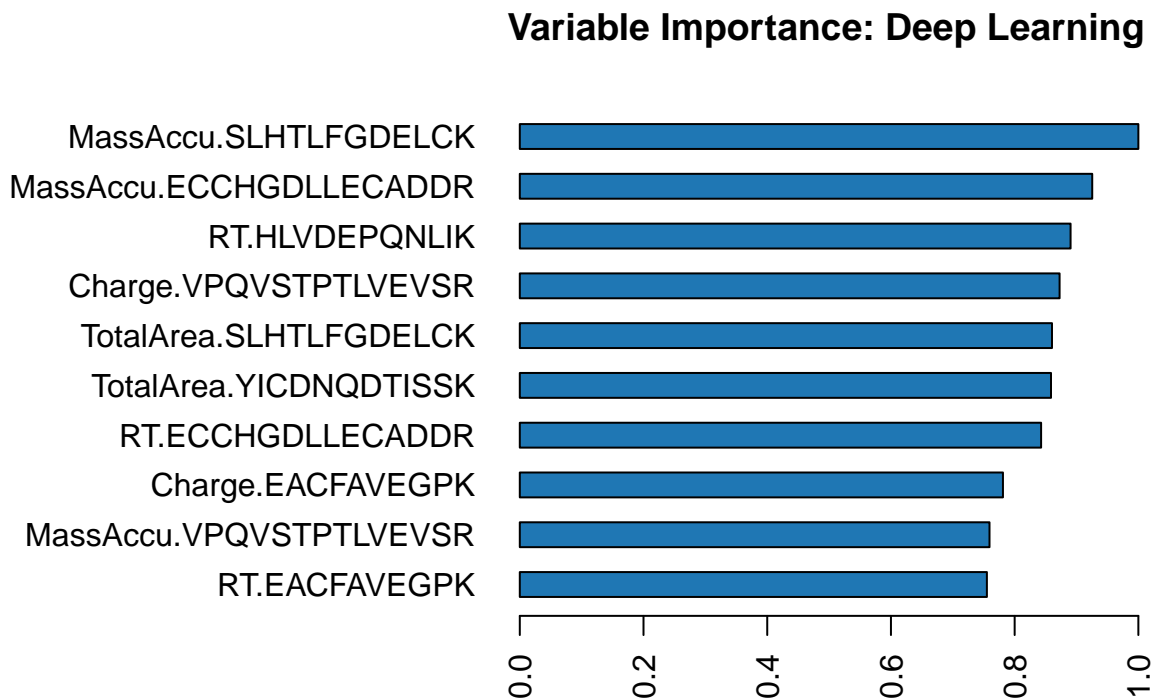
The training accuracy and testing accuracy decreases with increase in epochs.

5.0.3 Predictions on test data

```
dl_predict <- as.data.frame(h2o.predict(dl_model, test_h2o))
```

5.0.4 Variable importance

```
h2o.varimp_plot(dl_model)
```



5.0.5 Getting weights for neural network

```
weights1<- h2o.weights(dl_model,matrix_id = 1)  
print(head(weights1))
```

```
## RT.EACFAVEGPK MZ.EACFAVEGPK Charge.EACFAVEGPK TotalArea.EACFAVEGPK  
## 1 -0.23583737 -0.2797842 -0.30021629 -0.49712917  
## 2 -0.31172439 0.1067308 0.20294011 0.09682767  
## 3 0.15955730 -0.2717285 0.34951603 -0.13898845  
## 4 -0.16535486 -0.1180188 0.14910677 0.02682342  
## 5 0.04370809 -0.1684333 -0.05766352 0.04393569  
## MassAccu.EACFAVEGPK FWHM.EACFAVEGPK RT.ECCHGDLLECADDR MZ.ECCHGDLLECADDR  
## 1 -0.11241128 0.03293785 -0.3784887 -0.20930095  
## 2 0.16646102 -0.01113706 0.1443341 0.10131175  
## 3 -0.09741532 -0.24260718 -0.1864207 0.26914412
```

## 4	0.03393902	-0.22448434	-0.1328639	0.10982639
## 5	-0.01928469	-0.10195930	0.2842945	-0.07123732
##	Charge.ECCHGDLLECADDR	TotalArea.ECCHGDLLECADDR	MassAccu.ECCHGDLLECADDR	
## 1	-0.08028070		0.04380139	0.2390468
## 2	0.17654702		0.22064185	0.3332438
## 3	0.17712097		0.05794469	0.1121283
## 4	-0.09680744		-0.37012294	0.2191533
## 5	-0.25219941		-0.07897471	-0.2872470
##	FWHM.ECCHGDLLECADDR	RT.HLVDEPQNLIK	MZ.HLVDEPQNLIK	Charge.HLVDEPQNLIK
## 1	-0.177995548	0.2089175	-0.007526016	-0.25721249
## 2	-0.053703718	0.2001829	-0.103207536	0.25190017
## 3	0.002204252	0.3230875	-0.080076307	-0.04167950
## 4	0.239626080	-0.3683851	0.216559514	0.01580123
## 5	0.021929504	-0.1893954	0.080183007	0.11831298
##	TotalArea.HLVDEPQNLIK	MassAccu.HLVDEPQNLIK	FWHM.HLVDEPQNLIK	
## 1	0.20658407	-0.03116613	-0.01719454	
## 2	0.05558940	0.18113561	0.15477853	
## 3	-0.08160750	-0.11139227	-0.21041989	
## 4	-0.07090132	0.13873971	0.19957870	
## 5	0.17626619	-0.51180947	0.01717483	
##	RT.LVNELTEFAK	MZ.LVNELTEFAK	Charge.LVNELTEFAK	TotalArea.LVNELTEFAK
## 1	0.13477133	-0.10674237	0.09982824	0.11847905
## 2	0.25755301	-0.28599012	0.21880627	-0.07164115
## 3	-0.04675132	-0.14938121	-0.25736502	0.22915508
## 4	-0.18672875	-0.07407859	0.03334839	-0.13678196
## 5	0.32723492	-0.02658161	0.19788770	0.06873327
##	MassAccu.LVNELTEFAK	FWHM.LVNELTEFAK	RT.NECFLSHK	MZ.NECFLSHK
## 1	-0.09730984	-0.085679427	0.12883815	0.292881101
## 2	0.06985192	-0.299628794	0.01590926	0.146213681
## 3	-0.10182326	-0.224256024	-0.27157819	-0.231285438
## 4	-0.12174950	-0.109187134	-0.14838588	-0.178881302
## 5	-0.45769987	-0.001172399	0.05864862	0.005703649
##	Charge.NECFLSHK	TotalArea.NECFLSHK	MassAccu.NECFLSHK	FWHM.NECFLSHK
## 1	0.01242616	0.292510897	-0.1439769	0.01307709
## 2	0.12708127	0.047977872	-0.1019750	-0.18202820
## 3	0.12362847	0.091072686	-0.2498159	0.08090138
## 4	0.20265849	0.009479483	-0.1671181	-0.21013930
## 5	0.20184378	-0.055024203	0.1134884	0.19007911
##	RT.SLHTLFGDELCK	MZ.SLHTLFGDELCK	Charge.SLHTLFGDELCK	
## 1	-0.10318555	-0.134392232	-0.047550403	
## 2	-0.23059729	-0.153181806	0.036417175	
## 3	0.31246373	-0.005469696	0.003266339	
## 4	0.22693875	0.192572594	-0.317640662	
## 5	-0.04229377	0.291551530	-0.047417101	
##	TotalArea.SLHTLFGDELCK	MassAccu.SLHTLFGDELCK	FWHM.SLHTLFGDELCK	
## 1	-0.2143636	0.11644069	-0.1376150	
## 2	-0.3995810	-0.37309504	0.2236235	
## 3	0.0460235	0.09098575	0.1052235	
## 4	0.2126222	0.46947935	0.2968217	
## 5	0.1401082	-0.28656608	0.1849351	
##	RT.VPQVSTPTLVEVSR	MZ.VPQVSTPTLVEVSR	Charge.VPQVSTPTLVEVSR	
## 1	-0.35019854	-0.06006742	0.23780479	
## 2	-0.21153511	0.08133842	-0.29036209	
## 3	0.03918119	0.12063876	-0.35590869	

```

## 4      -0.19612859      0.09823170      -0.05662033
## 5      0.04685416      -0.01915597      0.29162684
##      TotalArea.VPQVSTPTLVEVSR MassAccu.VPQVSTPTLVEVSR FWHM.VPQVSTPTLVEVSR
## 1      0.22272693      -0.21007168      -0.04567311
## 2      0.13274157      -0.31248918      -0.04669969
## 3      0.16547132      -0.26304877      -0.35921317
## 4      -0.06847159      0.14238121      0.15663911
## 5      -0.28570506      -0.04939715      0.12096434
##      RT.YICDNQDTISSK MZ.YICDNQDTISSK Charge.YICDNQDTISSK
## 1      0.1365843      -0.19785093      0.27808824
## 2      -0.2237834      0.32785612      0.16589519
## 3      0.2327577      0.01206885      -0.21181442
## 4      0.2032687      -0.11087516      -0.04950723
## 5      0.1047876      0.04063049      0.05824081
##      TotalArea.YICDNQDTISSK MassAccu.YICDNQDTISSK FWHM.YICDNQDTISSK
## 1      -0.28742716      0.1310544      0.1410810
## 2      0.24357809      -0.2643718      -0.2426255
## 3      -0.04995213      -0.1191344      0.3911833
## 4      0.24030717      0.2192163      -0.1944355
## 5      -0.26405954      0.1498870      -0.1128908

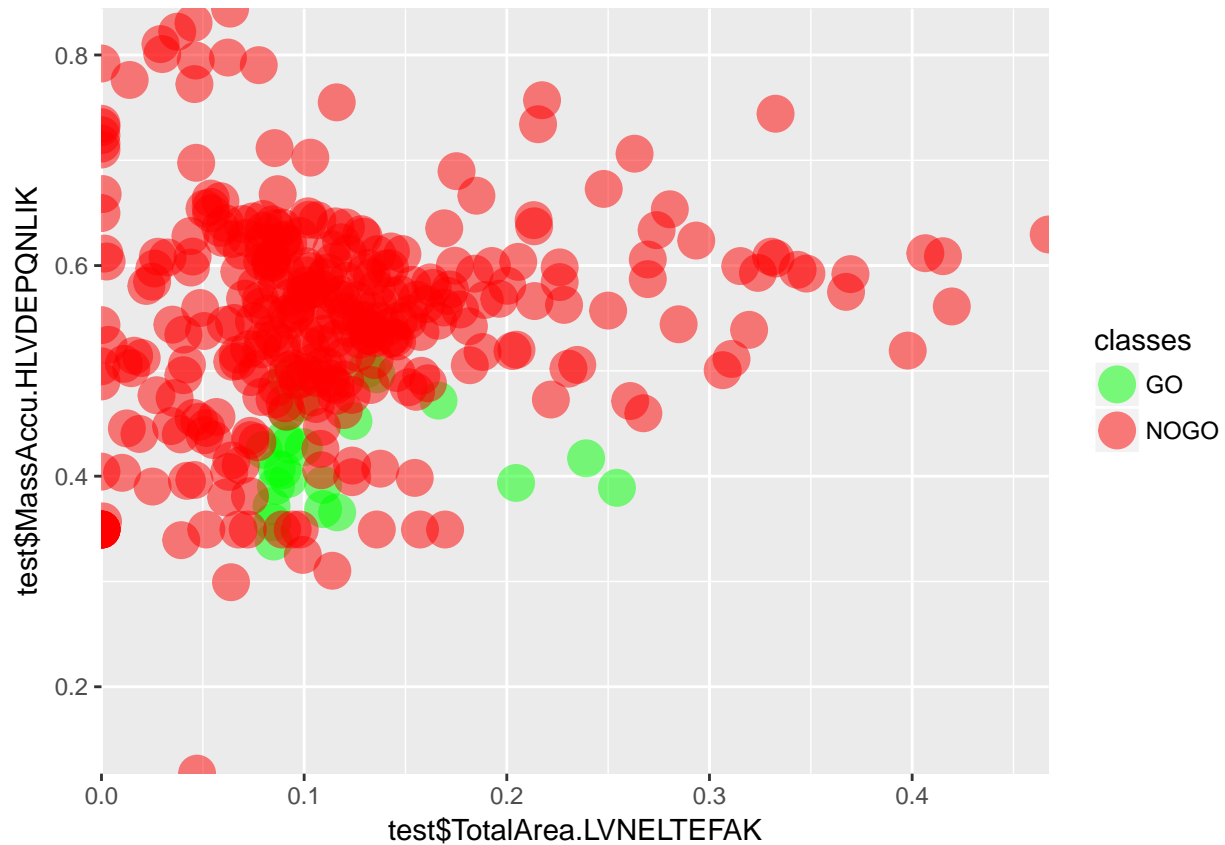
```

5.0.6 Plotting decision boundary for neural networks

```

Color <- c("GREEN","RED")
names(Color) <- c('GO','NOGO')
ggplot(data = test,aes(x = test$TotalArea.LVNELTEFAK, y = test$MassAccu.HLVDEPQNLIK)) +
  geom_point(aes(color = test$RESPONSE), size = 6, alpha = .5) +
  scale_colour_manual(name = 'classes', values = Color) +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0))

```



6 build the mlp(multi layer perceptron) deep learning model using h2o

```
set.seed(100)

dl_model2 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:48]),
  y= "RESPONSE",
  activation="Tanh",
  hidden=c(5,5),
  stopping_metric="mean_per_class_error",
  stopping_rounds = 5,
  stopping_tolerance=0.01,
  rate = 0.005, # Defaults to 0.005
  mini_batch_size = 1, # defaults to 1
  epochs=100,
  seed = 123, # give seed
  export_weights_and_biases = T, # export weights and biases defaults to false
  reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
```

```
summary(dl_model2)
```

```
## Model Details:
## =====
##
## H2OBinomialModel: deeplearning
## Model Key: dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossEntropy
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1     1     48   Input  0.00 %
## 2     2      5   Tanh  0.00 % 0.000000 0.000000  0.002646 0.002210
## 3     3      5   Tanh  0.00 % 0.000000 0.000000  0.001168 0.001267
## 4     4      2 Softmax      0.000000 0.000000  0.002892 0.000555
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000    0.014566    0.199495 -0.033308 0.046426
## 3 0.000000   -0.024504    0.494151  0.287324 0.076459
## 4 0.000000    0.140694    2.875058  0.000000 0.493199
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.02220305
## RMSE: 0.1490069
## LogLoss: 0.07940421
## Mean Per-Class Error: 0.1235652
## AUC:  0.9843856
## Gini: 0.9687713
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           GO NOGO   Error   Rate
## GO       63   20 0.240964   =20/83
## NOGO      6  967 0.006166   =6/973
## Totals 69  987 0.024621   =26/1056
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##
##           metric threshold   value idx
## 1           max f1  0.450857 0.986735 330
## 2           max f2  0.450857 0.990982 330
## 3           max f0point5 0.771159 0.987241 295
## 4           max accuracy 0.455604 0.975379 328
## 5           max precision 0.999989 1.000000  0
## 6           max recall  0.138910 1.000000 376
## 7           max specificity 0.999989 1.000000  0
## 8           max absolute_mcc 0.455604 0.820879 328
## 9   max min_per_class_accuracy 0.888299 0.945529 267
## 10  max mean_per_class_accuracy 0.904772 0.951095 260
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/>`
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
```

```

##
## MSE: 0.02081881
## RMSE: 0.1442873
## LogLoss: 0.07279464
## Mean Per-Class Error: 0.06705927
## AUC: 0.9864488
## Gini: 0.9728977
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      21      3 0.125000    =3/24
## NOGO      3    326 0.009119    =3/329
## Totals 24    329 0.016997    =6/353
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold      value idx
## 1      max f1 0.510167 0.990881 327
## 2      max f2 0.510167 0.990881 327
## 3      max f0point5 0.510167 0.990881 327
## 4      max accuracy 0.510167 0.983003 327
## 5      max precision 0.999993 1.000000 0
## 6      max recall 0.134955 1.000000 347
## 7      max specificity 0.999993 1.000000 0
## 8      max absolute_mcc 0.510167 0.865881 327
## 9      max min_per_class_accuracy 0.865585 0.951368 312
## 10     max mean_per_class_accuracy 0.865585 0.954851 312
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/I>)`
##
##
## Scoring History:
##      timestamp      duration training_speed      epochs iterations
## 1 2018-06-05 11:22:19 0.000 sec              0.00000      0
## 2 2018-06-05 11:22:19 0.212 sec    34064 obs/sec 1.00000      1
## 3 2018-06-05 11:22:19 0.282 sec    29746 obs/sec 2.00000      2
## 4 2018-06-05 11:22:19 0.332 sec    30461 obs/sec 3.00000      3
## 5 2018-06-05 11:22:19 0.392 sec    29333 obs/sec 4.00000      4
##      samples training_rmse training_logloss training_auc training_lift
## 1      0.000000
## 2 1056.000000      0.28237      0.28231      0.82289      0.98664
## 3 2112.000000      0.23118      0.18770      0.91223      1.08530
## 4 3168.000000      0.20979      0.15442      0.93608      1.08530
## 5 4224.000000      0.19559      0.13393      0.94915      1.08530
##      training_classification_error validation_rmse validation_logloss
## 1
## 2      0.07860      0.27834      0.25853
## 3      0.07008      0.23571      0.17472
## 4      0.05682      0.21349      0.14398
## 5      0.04830      0.19823      0.12551
##      validation_auc validation_lift validation_classification_error
## 1
## 2      0.83169      1.07295      0.06516
## 3      0.91958      1.07295      0.06516
## 4      0.94453      1.07295      0.05949

```

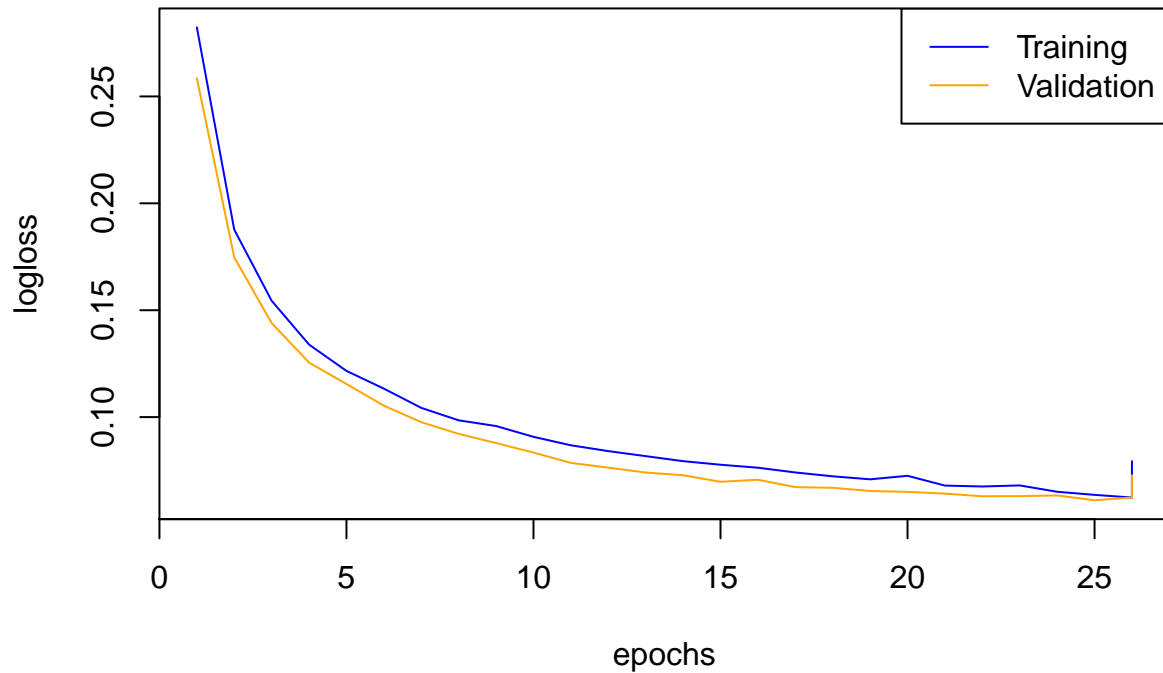
```

## 5          0.95517          1.07295          0.05099
##
## ---
##          timestamp    duration training_speed    epochs iterations
## 23 2018-06-05 11:22:20  1.336 sec  33767 obs/sec 22.00000        22
## 24 2018-06-05 11:22:20  1.377 sec  34016 obs/sec 23.00000        23
## 25 2018-06-05 11:22:20  1.424 sec  33973 obs/sec 24.00000        24
## 26 2018-06-05 11:22:21  1.462 sec  34375 obs/sec 25.00000        25
## 27 2018-06-05 11:22:21  1.520 sec  34362 obs/sec 26.00000        26
## 28 2018-06-05 11:22:21  1.562 sec  33688 obs/sec 26.00000        26
##          samples training_rmse training_logloss training_auc training_lift
## 23 23232.000000          0.13724          0.06755          0.98882          1.08530
## 24 24288.000000          0.13779          0.06805          0.98912          1.08530
## 25 25344.000000          0.13530          0.06507          0.98995          1.08530
## 26 26400.000000          0.13294          0.06364          0.99016          1.08530
## 27 27456.000000          0.13219          0.06239          0.99060          1.08530
## 28 27456.000000          0.14901          0.07940          0.98439          1.08530
##          training_classification_error validation_rmse validation_logloss
## 23          0.02178          0.13439          0.06297
## 24          0.02178          0.13462          0.06304
## 25          0.02083          0.13604          0.06335
## 26          0.02083          0.13296          0.06112
## 27          0.01894          0.13525          0.06236
## 28          0.02462          0.14429          0.07279
##          validation_auc validation_lift validation_classification_error
## 23          0.99012          1.07295          0.01983
## 24          0.99050          1.07295          0.01983
## 25          0.98999          1.07295          0.01983
## 26          0.99101          1.07295          0.01983
## 27          0.99012          1.07295          0.01983
## 28          0.98645          1.07295          0.01700
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
##          variable relative_importance scaled_importance percentage
## 1   Charge.VPQVSTPTLVEVSR          1.000000          1.000000  0.034240
## 2   MassAccu.HLVDEPQNLIK          0.985779          0.985779  0.033753
## 3 MassAccu.ECCHGDLLECADDR          0.950572          0.950572  0.032548
## 4   TotalArea.EACFAVEGPK          0.910523          0.910523  0.031177
## 5       RT.HLVDEPQNLIK          0.906599          0.906599  0.031042
##
## ---
##          variable relative_importance scaled_importance percentage
## 43  FWHM.VPQVSTPTLVEVSR          0.344528          0.344528  0.011797
## 44  TotalArea.NECFLSHK          0.340728          0.340728  0.011667
## 45  Charge.SLHTLFGDELCK          0.319727          0.319727  0.010948
## 46  MZ.VPQVSTPTLVEVSR          0.309764          0.309764  0.010606
## 47  FWHM.ECCHGDLLECADDR          0.267567          0.267567  0.009162
## 48 MassAccu.YICDNQDTISSK          0.236624          0.236624  0.008102

```

```
plot(dl_model2)
```

Scoring History



6.0.1 shutdown h2o

6.1 Main Data

```
head(Train)
```

```
##      idfile                                     PepSeq
## 1  14811                                     EAC(Carbamidomethyl)FAVEGPK
## 2  14811 EC(Carbamidomethyl)C(Carbamidomethyl)HGDLLEC(Carbamidomethyl)ADDR
## 4  14811                                     HLVDEPQNLIK
## 5  14811                                     LVNELTEFAK
## 6  14811                                     NEC(Carbamidomethyl)FLSHK
## 7  14811 SLHTLFGDEL(Carbamidomethyl)K
##      RT      MZ Charge TotalArea  MassAccu  FWHM
## 1 734.913 554.261     2  2.36e+08 -0.0652368 2.04135
## 2 715.251 583.893     3  1.88e+08  0.2566130 1.90107
## 4 739.787 653.362     2  9.38e+08  0.1110990 1.97179
## 5 792.170 582.319     2  6.85e+08  0.0907428 1.67105
## 6 675.845 517.740     2  3.88e+08  0.0501789 1.95090
## 7 777.652 710.350     2  7.87e+07 -0.4440630 1.83461
```

```
nrow(Train)
```

```
## [1] 856
```



```
Train %>% group_by(PepSeq) %>% summarise(Count = n())
```

```
## # A tibble: 8 x 2
##   PepSeq                                Count
##   <fct>                                <int>
## 1 EAC(Carbamidomethyl)FAVEGPK                107
## 2 EC(Carbamidomethyl)C(Carbamidomethyl)HGDLLEC(Carbamidomethyl)ADDR 107
## 3 HLVDEPQNLIK                                107
## 4 LVNELTEFAK                                107
## 5 NEC(Carbamidomethyl)FLSHK                  107
## 6 SLHTLFGDEL(Carbamidomethyl)K              107
## 7 VPQVSTPTLVEVSR                            107
## 8 YIC(Carbamidomethyl)DNQDTISSK             107
```

7 Simulation : Spike in mean X 10 per feature to check variable importance

```
spike_mean <- function(num_col,value){
  # Train Data
  #one peptide LVNELTEFAK

  #generate multivariate normal data
  #parameters from a training sample
  n<-100 #incontrol observations
  m<-100 #ooc observations

  # Simulating in Control data with n observations
  mean <-c(with(data=Train,tapply(RT,INDEX=PepSeq,FUN=mean))[5],
           with(data=Train,tapply(TotalArea,INDEX=PepSeq,FUN=mean))[5],
           with(data=Train,tapply(MassAccu,INDEX=PepSeq,FUN=mean))[5],
           with(data=Train,tapply(FWHM,INDEX=PepSeq,FUN=mean))[5]
           )
  covar<- cov(Train[Train$PepSeq=="LVNELTEFAK",c(3,6,7,8)])

  Sim_ic_1 <-data.frame(idfile=1:n,PepSeq=rep("LVNELTEFAK",n),mvrnorm(n, mean, covar))
  colnames(Sim_ic_1)<-c("idfile","PepSeq","RT","TotalArea","MassAccu","FWHM")

  RESPONSE <- c("GO")
  Sim_ic_1 <- cbind(Sim_ic_1,RESPONSE) # simulation effect 1 incontrol observation

  # Simulating Out of Control data with m observations
  # Sim_oc_1 <-data.frame(idfile=(n+1):(n+m),PepSeq=rep("LVNELTEFAK",m),
  #                       mvrnorm(m, mean+(10*c(covar[1,1],1.0*covar[2,2],3.0*covar[3,3],1.0*covar[4,4])),
  #                       covar))
  if(num_col == 1)
    Sim_oc_1 <- data.frame(idfile=(n+1):(n+m),PepSeq=rep("LVNELTEFAK",m), # increase in mean FWHM X 10
                          mvrnorm(m, mean*c(value,1,1,1),
                          covar))
```

```

else if(num_col == 2)
  Sim_oc_1 <- data.frame(idfile=(n+1):(n+m),PepSeq=rep("LVNELTEFAK",m), # increase in mean FWHM X 10
                        mvrnorm(m, mean*c(1,value,1,1),
                                covar))
else if(num_col == 3)
  Sim_oc_1 <- data.frame(idfile=(n+1):(n+m),PepSeq=rep("LVNELTEFAK",m), # increase in mean FWHM X 10
                        mvrnorm(m, mean*c(1,1,value,1),
                                covar))
else
  Sim_oc_1 <- data.frame(idfile=(n+1):(n+m),PepSeq=rep("LVNELTEFAK",m), # increase in mean FWHM X 10
                        mvrnorm(m, mean*c(1,1,1,value),
                                covar))

colnames(Sim_oc_1) <- c("idfile","PepSeq","RT","TotalArea","MassAccu","FWHM")

RESPONSE <- c("NOGO")
Sim_oc_1 <- cbind(Sim_oc_1,RESPONSE)
new_data <- rbind(Sim_ic_1,Sim_oc_1)
return(new_data)
}

new_data <- spike_mean(1,10)
maxs <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, max)
mins <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, min)

scaled_data <- as.data.frame(scale(new_data %>% dplyr::select(-c(idfile,PepSeq,RESPONSE)), center = mins,
#scaled_data$RESPONSE <- ifelse(new_data$RESPONSE == "GO",1,0)
scaled_data$RESPONSE <- as.factor(new_data$RESPONSE)
#scaled_data$RESPONSE <- as.factor(scaled_data$RESPONSE)
scaled_data$idfile <- new_data$idfile

#select random ind for train and test
set.seed(123)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(scaled_data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(scaled_data)), size = smp_size)

train_sim1 <- scaled_data[train_ind,]
test_sim1 <- scaled_data[-train_ind,]

#Building Neural Network
library(h2o)
#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

## Connection successful!

```

```
##
## R is connected to the H2O cluster:
##   H2O cluster uptime:      16 minutes 33 seconds
##   H2O cluster timezone:    America/New_York
##   H2O data parsing timezone: UTC
##   H2O cluster version:     3.18.0.8
##   H2O cluster version age:  1 month and 16 days
##   H2O cluster name:        H2O_started_from_R_Shantam_Gupta_kyo754
##   H2O cluster total nodes:  1
##   H2O cluster total memory: 2.63 GB
##   H2O cluster total cores:  4
##   H2O cluster allowed cores: 4
##   H2O cluster healthy:      TRUE
##   H2O Connection ip:        localhost
##   H2O Connection port:      54321
##   H2O Connection proxy:     NA
##   H2O Internal Security:     FALSE
##   H2O API Extensions:       Algos, AutoML, Core V3, Core V4
##   R Version:                 R version 3.3.3 (2017-03-06)
```

```
#import r objects to h2o cloud
train_h2o <- as.h2o(train_sim1)
test_h2o <- as.h2o(test_sim1)

set.seed(100)

dl_model_sim1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:4]),
  y= "RESPONSE",
  activation="Tanh",
  hidden=c(2,2),
  stopping_metric="mean_per_class_error",
  stopping_rounds = 5,
  stopping_tolerance=0.001,
  rate = 0.005, # Defaults to 0.005
  mini_batch_size = 1,# defaults to 1
  epochs=100,
  seed = 123, # give seed
  export_weights_and_biases = T, # export weights and biases defaults to false
  reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
summary(dl_model_sim1)
```

```
## Model Details:
```

```
## =====
```

```
##
```

```
## H2OBinomialModel: deeplearning
```

```
## Model Key: dl_model_first
```

```
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossEntropy
batch size 1
```

```
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1      4   Input  0.00 %
```

```

## 2      2      2      Tanh  0.00 % 0.000000 0.000000  0.002463 0.001915
## 3      3      2      Tanh  0.00 % 0.000000 0.000000  0.101875 0.102434
## 4      4      2  Softmax      0.000000 0.000000  0.006101 0.004078
##      momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000 -0.408571  0.598224 -0.009116 0.012083
## 3 0.000000  0.723658  1.104087  0.014922 0.024276
## 4 0.000000 -2.140051  3.418088  0.000000 0.010361
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.01541795
## RMSE:  0.124169
## LogLoss:  0.07021577
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      74      0 0.000000  =0/74
## NOGO      0      76 0.000000  =0/76
## Totals 74      76 0.000000  =0/150
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold      value idx
## 1      max f1  0.574902 1.000000  75
## 2      max f2  0.574902 1.000000  75
## 3      max f0point5 0.574902 1.000000  75
## 4      max accuracy 0.574902 1.000000  75
## 5      max precision 0.995948 1.000000  0
## 6      max recall  0.574902 1.000000  75
## 7      max specificity 0.995948 1.000000  0
## 8      max absolute_mcc 0.574902 1.000000  75
## 9      max min_per_class_accuracy 0.574902 1.000000  75
## 10 max mean_per_class_accuracy 0.574902 1.000000  75
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/I>`
## H2OBinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.008488261
## RMSE:  0.09213176
## LogLoss:  0.05680938
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      26      0 0.000000  =0/26

```

```

## NOGO      0    24 0.000000  =0/24
## Totals 26    24 0.000000  =0/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##
##           metric threshold   value idx
## 1           max f1  0.849627 1.000000 23
## 2           max f2  0.849627 1.000000 23
## 3           max f0point5 0.849627 1.000000 23
## 4           max accuracy 0.849627 1.000000 23
## 5           max precision 0.996744 1.000000  0
## 6           max recall  0.849627 1.000000 23
## 7           max specificity 0.996744 1.000000  0
## 8           max absolute_mcc 0.849627 1.000000 23
## 9   max min_per_class_accuracy 0.849627 1.000000 23
## 10 max mean_per_class_accuracy 0.849627 1.000000 23
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##           timestamp   duration training_speed   epochs iterations
## 1  2018-06-05 11:22:23  0.000 sec           0.00000 0
## 2  2018-06-05 11:22:23  0.006 sec 150000 obs/sec 1.00000 1
## 3  2018-06-05 11:22:23  0.006 sec 300000 obs/sec 2.00000 2
## 4  2018-06-05 11:22:23  0.053 sec  26470 obs/sec 3.00000 3
## 5  2018-06-05 11:22:23  0.053 sec  35294 obs/sec 4.00000 4
## 6  2018-06-05 11:22:23  0.069 sec  44117 obs/sec 5.00000 5
## 7  2018-06-05 11:22:23  0.069 sec  52941 obs/sec 6.00000 6
## 8  2018-06-05 11:22:23  0.084 sec  32812 obs/sec 7.00000 7
## 9  2018-06-05 11:22:23  0.100 sec  37500 obs/sec 8.00000 8
## 10 2018-06-05 11:22:23  0.106 sec  35526 obs/sec 9.00000 9
## 11 2018-06-05 11:22:23  0.122 sec  27777 obs/sec 10.00000 10
## 12 2018-06-05 11:22:23  0.153 sec  30555 obs/sec 11.00000 11
## 13 2018-06-05 11:22:23  0.153 sec  33333 obs/sec 12.00000 12
## 14 2018-06-05 11:22:23  0.169 sec  36111 obs/sec 13.00000 13
##
##           samples training_rmse training_logloss training_auc training_lift
## 1           0.000000
## 2    150.000000      0.51282      0.92573      0.73471      1.97368
## 3    300.000000      0.47114      0.73804      0.78841      1.97368
## 4    450.000000      0.42235      0.56635      0.85188      1.97368
## 5    600.000000      0.37093      0.42528      0.90807      1.97368
## 6    750.000000      0.32329      0.32105      0.94488      1.97368
## 7    900.000000      0.28263      0.24797      0.96871      1.97368
## 8   1050.000000      0.24851      0.19671      0.98204      1.97368
## 9   1200.000000      0.21949      0.15974      0.98826      1.97368
## 10  1350.000000      0.19487      0.13251      0.99307      1.97368
## 11  1500.000000      0.17370      0.11167      0.99449      1.97368
## 12  1650.000000      0.15499      0.09491      0.99716      1.97368
## 13  1800.000000      0.13851      0.08131      0.99876      1.97368
## 14  1950.000000      0.12417      0.07022      1.00000      1.97368
##
##           training_classification_error validation_rmse validation_logloss
## 1
## 2           0.33333      0.54310      0.99967
## 3           0.28667      0.49791      0.78645

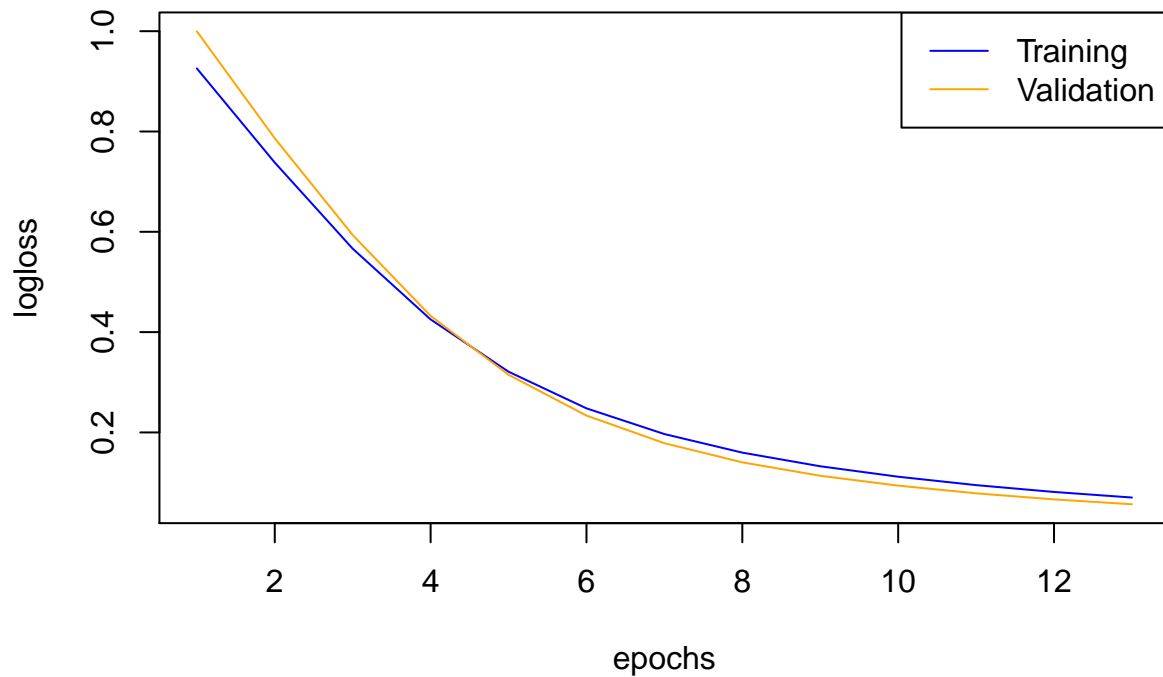
```

```

## 4          0.22667          0.44524          0.59359
## 5          0.17333          0.38546          0.43234
## 6          0.13333          0.32573          0.31490
## 7          0.11333          0.27222          0.23361
## 8          0.08000          0.22776          0.17848
## 9          0.06000          0.19166          0.14033
## 10         0.03333          0.16306          0.11355
## 11         0.03333          0.14021          0.09392
## 12         0.02000          0.12113          0.07858
## 13         0.01333          0.10527          0.06647
## 14         0.00000          0.09213          0.05681
## validation_auc validation_lift validation_classification_error
## 1
## 2          0.75641          2.08333          0.32000
## 3          0.80449          2.08333          0.28000
## 4          0.86218          2.08333          0.24000
## 5          0.91987          2.08333          0.16000
## 6          0.96955          2.08333          0.08000
## 7          0.99359          2.08333          0.04000
## 8          0.99840          2.08333          0.02000
## 9          1.00000          2.08333          0.00000
## 10         1.00000          2.08333          0.00000
## 11         1.00000          2.08333          0.00000
## 12         1.00000          2.08333          0.00000
## 13         1.00000          2.08333          0.00000
## 14         1.00000          2.08333          0.00000
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
## variable relative_importance scaled_importance percentage
## 1      RT          1.000000          1.000000  0.401582
## 2  MassAccu          0.724706          0.724706  0.291029
## 3      FWHM          0.510556          0.510556  0.205031
## 4 TotalArea          0.254886          0.254886  0.102358
plot(dl_model_sim1)

```

Scoring History



```
new_data <- spike_mean(2,10)
maxs <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, max)
mins <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, min)

scaled_data <- as.data.frame(scale(new_data %>% dplyr::select(-c(idfile,PepSeq,RESPONSE)), center = min,
#scaled_data$RESPONSE <- ifelse(new_data$RESPONSE == "GO",1,0)
scaled_data$RESPONSE <- as.factor(new_data$RESPONSE)
#scaled_data$RESPONSE <- as.factor(scaled_data$RESPONSE)
scaled_data$idfile <- new_data$idfile

#select random ind for train and test
set.seed(123)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(scaled_data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(scaled_data)), size = smp_size)

train_sim1 <- scaled_data[train_ind,]
test_sim1 <- scaled_data[-train_ind,]

#Building Neural Network
library(h2o)
```

```

#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

## Connection successful!
##
## R is connected to the H2O cluster:
##   H2O cluster uptime:      16 minutes 36 seconds
##   H2O cluster timezone:    America/New_York
##   H2O data parsing timezone: UTC
##   H2O cluster version:     3.18.0.8
##   H2O cluster version age:  1 month and 16 days
##   H2O cluster name:        H2O_started_from_R_Shantam_Gupta_kyo754
##   H2O cluster total nodes:  1
##   H2O cluster total memory: 2.63 GB
##   H2O cluster total cores:  4
##   H2O cluster allowed cores: 4
##   H2O cluster healthy:      TRUE
##   H2O Connection ip:        localhost
##   H2O Connection port:      54321
##   H2O Connection proxy:     NA
##   H2O Internal Security:     FALSE
##   H2O API Extensions:        Algos, AutoML, Core V3, Core V4
##   R Version:                 R version 3.3.3 (2017-03-06)

#import r objects to h2o cloud
train_h2o <- as.h2o(train_sim1)
test_h2o <- as.h2o(test_sim1)

set.seed(100)

dl_model_sim1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:4]),
  y= "RESPONSE",
  activatio="Tanh",
  hidden=c(2,2),
  stopping_metric="mean_per_class_error",
  stopping_rounds = 5,
  stopping_tolerance=0.001,
  rate = 0.005, # Defaults to 0.005
  mini_batch_size = 1,# defaults to 1
  epochs=100,
  seed = 123, # give seed
  export_weights_and_biases = T, # export weights and biases defaults to false
  reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
summary(dl_model_sim1)

## Model Details:
## =====

```



```

##
## H20BinomialModel: deeplearning
## Model Key: dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossE
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1      4   Input  0.00 %
## 2      2      2   Tanh  0.00 % 0.000000 0.000000  0.003115 0.003149
## 3      3      2   Tanh  0.00 % 0.000000 0.000000  0.103662 0.106827
## 4      4      2 Softmax      0.000000 0.000000  0.004898 0.003095
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000  -0.411499   0.646284 -0.026006 0.024857
## 3 0.000000   0.788713   1.074335  0.009208 0.023556
## 4 0.000000  -2.140051   3.415698 -0.000000 0.017427
##
## H20BinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.01960665
## RMSE:  0.1400237
## LogLoss:  0.08526118
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           GO NOGO      Error      Rate
## GO         74    0 0.000000   =0/74
## NOGO        0   76 0.000000   =0/76
## Totals 74   76 0.000000   =0/150
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##           metric threshold   value idx
## 1           max f1  0.607347 1.000000  75
## 2           max f2  0.607347 1.000000  75
## 3           max f0point5 0.607347 1.000000  75
## 4           max accuracy 0.607347 1.000000  75
## 5           max precision 0.996383 1.000000   0
## 6           max recall  0.607347 1.000000  75
## 7           max specificity 0.996383 1.000000   0
## 8           max absolute_mcc 0.607347 1.000000  75
## 9   max min_per_class_accuracy 0.607347 1.000000  75
## 10 max mean_per_class_accuracy 0.607347 1.000000  75
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H20BinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.01650105
## RMSE:  0.1284564
## LogLoss:  0.07390136

```

```

## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error   Rate
## GO      26    0 0.000000  =0/26
## NOGO     0   24 0.000000  =0/24
## Totals 26   24 0.000000  =0/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold   value idx
## 1      max f1  0.900900 1.000000  23
## 2      max f2  0.900900 1.000000  23
## 3      max f0point5 0.900900 1.000000  23
## 4      max accuracy 0.900900 1.000000  23
## 5      max precision 0.996023 1.000000   0
## 6      max recall  0.900900 1.000000  23
## 7      max specificity 0.996023 1.000000   0
## 8      max absolute_mcc 0.900900 1.000000  23
## 9  max min_per_class_accuracy 0.900900 1.000000  23
## 10 max mean_per_class_accuracy 0.900900 1.000000  23
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##      timestamp      duration training_speed   epochs iterations
## 1  2018-06-05 11:22:25  0.000 sec           0.000000         0
## 2  2018-06-05 11:22:25  0.033 sec    25000 obs/sec    1.000000         1
## 3  2018-06-05 11:22:26  0.049 sec    13636 obs/sec    2.000000         2
## 4  2018-06-05 11:22:26  0.064 sec    12162 obs/sec    3.000000         3
## 5  2018-06-05 11:22:26  0.080 sec    11320 obs/sec    4.000000         4
## 6  2018-06-05 11:22:26  0.134 sec     7575 obs/sec    5.000000         5
## 7  2018-06-05 11:22:26  0.152 sec     8411 obs/sec    6.000000         6
## 8  2018-06-05 11:22:26  0.164 sec     9633 obs/sec    7.000000         7
## 9  2018-06-05 11:22:26  0.182 sec    10084 obs/sec    8.000000         8
## 10 2018-06-05 11:22:26  0.194 sec    10465 obs/sec    9.000000         9
## 11 2018-06-05 11:22:26  0.214 sec    10791 obs/sec   10.000000        10
##
##      samples training_rmse training_logloss training_auc training_lift
## 1      0.000000
## 2    150.000000      0.55022      1.08180      0.67710      1.97368
## 3    300.000000      0.49915      0.81487      0.76618      1.97368
## 4    450.000000      0.43912      0.58269      0.84780      1.97368
## 5    600.000000      0.37585      0.41091      0.91430      1.97368
## 6    750.000000      0.31580      0.29276      0.95110      1.97368
## 7    900.000000      0.26485      0.21578      0.97173      1.97368
## 8   1050.000000      0.22307      0.16440      0.98506      1.97368
## 9   1200.000000      0.18920      0.12895      0.99413      1.97368
## 10  1350.000000      0.16215      0.10391      0.99680      1.97368
## 11  1500.000000      0.14002      0.08526      1.00000      1.97368
##
##      training_classification_error validation_rmse validation_logloss
## 1
## 2      0.42667      0.54451      0.98841

```

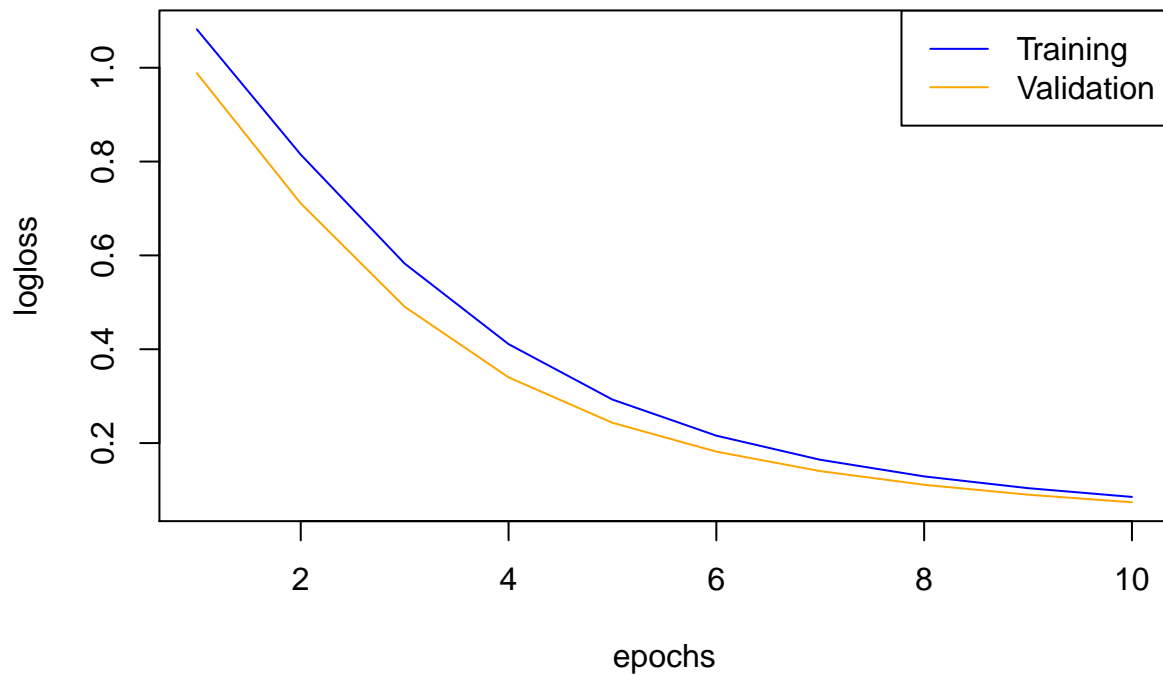
```

## 3          0.34000          0.47942          0.71093
## 4          0.27333          0.40275          0.49068
## 5          0.18667          0.33205          0.34009
## 6          0.12000          0.27604          0.24326
## 7          0.08667          0.23371          0.18177
## 8          0.06000          0.19973          0.14025
## 9          0.02667          0.17167          0.11110
## 10         0.02000          0.14833          0.08998
## 11         0.00000          0.12846          0.07390
## validation_auc validation_lift validation_classification_error
## 1
## 2          0.73718          2.08333          0.38000
## 3          0.82372          2.08333          0.30000
## 4          0.89744          2.08333          0.24000
## 5          0.95513          2.08333          0.12000
## 6          0.97756          2.08333          0.08000
## 7          0.99679          2.08333          0.02000
## 8          1.00000          2.08333          0.00000
## 9          1.00000          2.08333          0.00000
## 10         1.00000          2.08333          0.00000
## 11         1.00000          2.08333          0.00000
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
## variable relative_importance scaled_importance percentage
## 1 TotalArea          1.000000          1.000000  0.406007
## 2 MassAccu           0.690363          0.690363  0.280292
## 3 FWHM               0.517246          0.517246  0.210006
## 4 RT                 0.255402          0.255402  0.103695

```

```
plot(dl_model_sim1)
```

Scoring History



```
new_data <- spike_mean(3,10)
maxs <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, max)
mins <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, min)

scaled_data <- as.data.frame(scale(new_data %>% dplyr::select(-c(idfile,PepSeq,RESPONSE)), center = mins,
#scaled_data$RESPONSE <- ifelse(new_data$RESPONSE == "GO",1,0)
scaled_data$RESPONSE <- as.factor(new_data$RESPONSE)
#scaled_data$RESPONSE <- as.factor(scaled_data$RESPONSE)
scaled_data$idfile <- new_data$idfile

#select random ind for train and test
set.seed(123)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(scaled_data))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(scaled_data)), size = smp_size)

train_sim1 <- scaled_data[train_ind,]
test_sim1 <- scaled_data[-train_ind,]

#Building Neural Network
library(h2o)
```

```

#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

## Connection successful!
##
## R is connected to the H2O cluster:
##   H2O cluster uptime:      16 minutes 38 seconds
##   H2O cluster timezone:    America/New_York
##   H2O data parsing timezone: UTC
##   H2O cluster version:     3.18.0.8
##   H2O cluster version age:  1 month and 16 days
##   H2O cluster name:        H2O_started_from_R_Shantam_Gupta_kyo754
##   H2O cluster total nodes:  1
##   H2O cluster total memory: 2.63 GB
##   H2O cluster total cores:  4
##   H2O cluster allowed cores: 4
##   H2O cluster healthy:      TRUE
##   H2O Connection ip:        localhost
##   H2O Connection port:      54321
##   H2O Connection proxy:     NA
##   H2O Internal Security:    FALSE
##   H2O API Extensions:       Algos, AutoML, Core V3, Core V4
##   R Version:                R version 3.3.3 (2017-03-06)

#import r objects to h2o cloud
train_h2o <- as.h2o(train_sim1)
test_h2o <- as.h2o(test_sim1)

set.seed(100)

dl_model_sim1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:4]),
  y= "RESPONSE",
  activatio="Tanh",
  hidden=c(2,2),
  stopping_metric="mean_per_class_error",
  stopping_rounds = 5,
  stopping_tolerance=0.001,
  rate = 0.005, # Defaults to 0.005
  mini_batch_size = 1,# defaults to 1
  epochs=100,
  seed = 123, # give seed
  export_weights_and_biases = T, # export weights and biases defaults to false
  reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
summary(dl_model_sim1)

## Model Details:
## =====

```

```

##
## H20BinomialModel: deeplearning
## Model Key: dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossE
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1      4   Input  0.00 %
## 2      2      2   Tanh  0.00 % 0.000000 0.000000  0.006331 0.005909
## 3      3      2   Tanh  0.00 % 0.000000 0.000000  0.096842 0.098050
## 4      4      2 Softmax      0.000000 0.000000  0.001924 0.000296
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000   -0.327752   0.676920 -0.023756 0.015411
## 3 0.000000    0.980989   1.081806 -0.057519 0.053647
## 4 0.000000   -2.140051   3.382374 -0.000000 0.004583
##
## H20BinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.0557924
## RMSE:  0.2362042
## LogLoss:  0.230534
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           GO NOGO      Error      Rate
## GO         74    0 0.000000   =0/74
## NOGO        0   76 0.000000   =0/76
## Totals 74    76 0.000000   =0/150
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##           metric threshold   value idx
## 1           max f1  0.560585 1.000000  75
## 2           max f2  0.560585 1.000000  75
## 3           max f0point5 0.560585 1.000000  75
## 4           max accuracy 0.560585 1.000000  75
## 5           max precision 0.979840 1.000000   0
## 6           max recall  0.560585 1.000000  75
## 7           max specificity 0.979840 1.000000   0
## 8           max absolute_mcc 0.560585 1.000000  75
## 9   max min_per_class_accuracy 0.560585 1.000000  75
## 10 max mean_per_class_accuracy 0.560585 1.000000  75
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H20BinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE:  0.03796815
## RMSE:  0.1948542
## LogLoss:  0.1820889

```

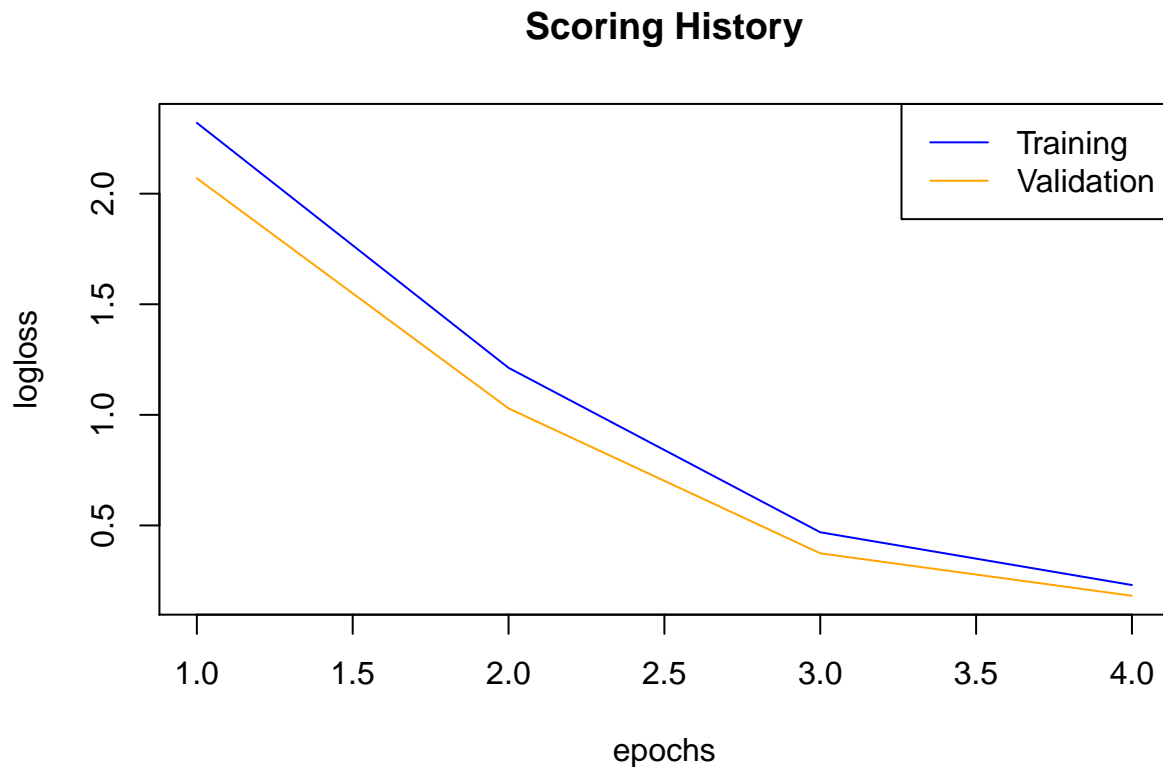
```

## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error   Rate
## GO      26    0 0.000000  =0/26
## NOGO     0   24 0.000000  =0/24
## Totals 26   24 0.000000  =0/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold   value idx
## 1      max f1  0.618661 1.000000  23
## 2      max f2  0.618661 1.000000  23
## 3      max f0point5 0.618661 1.000000  23
## 4      max accuracy 0.618661 1.000000  23
## 5      max precision 0.980413 1.000000   0
## 6      max recall  0.618661 1.000000  23
## 7      max specificity 0.980413 1.000000   0
## 8      max absolute_mcc 0.618661 1.000000  23
## 9  max min_per_class_accuracy 0.618661 1.000000  23
## 10 max mean_per_class_accuracy 0.618661 1.000000  23
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##      timestamp   duration training_speed  epochs iterations
## 1 2018-06-05 11:22:28 0.000 sec           0.000000      0
## 2 2018-06-05 11:22:28 0.037 sec 150000 obs/sec 1.000000      1
## 3 2018-06-05 11:22:28 0.069 sec 17647 obs/sec 2.000000      2
## 4 2018-06-05 11:22:28 0.069 sec 26470 obs/sec 3.000000      3
## 5 2018-06-05 11:22:28 0.084 sec 18750 obs/sec 4.000000      4
##      samples training_rmse training_logloss training_auc training_lift
## 1 0.000000
## 2 150.000000 0.83951 2.32023 0.07913 0.00000
## 3 300.000000 0.66654 1.21269 0.31294 0.98684
## 4 450.000000 0.39809 0.46915 0.84495 1.97368
## 5 600.000000 0.23620 0.23053 1.00000 1.97368
##      training_classification_error validation_rmse validation_logloss
## 1
## 2 0.49333 0.80521 2.06955
## 3 0.46000 0.62335 1.02920
## 4 0.21333 0.34090 0.37388
## 5 0.00000 0.19485 0.18209
##      validation_auc validation_lift validation_classification_error
## 1
## 2 0.21474 0.00000 0.52000
## 3 0.41346 2.08333 0.48000
## 4 0.94551 2.08333 0.16000
## 5 1.00000 2.08333 0.00000
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====

```

```
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1  MassAccu           1.000000           1.000000    0.461504
## 2    FWHM            0.516957           0.516957    0.238578
## 3     RT            0.388668           0.388668    0.179372
## 4 TotalArea          0.261205           0.261205    0.120547
```

```
plot(dl_model_sim1)
```



```
new_data <- spike_mean(4,10)
maxs <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, max)
mins <- apply(new_data %>% dplyr::select(-c(idfile,PepSeq, RESPONSE)), 2, min)

scaled_data <- as.data.frame(scale(new_data %>% dplyr::select(-c(idfile,PepSeq,RESPONSE)), center = mins,
#scaled_data$RESPONSE <- ifelse(new_data$RESPONSE == "GO",1,0)
scaled_data$RESPONSE <- as.factor(new_data$RESPONSE)
#scaled_data$RESPONSE <- as.factor(scaled_data$RESPONSE)
scaled_data$idfile <- new_data$idfile

#select random ind for train and test
set.seed(123)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(scaled_data))

## set the seed to make your partition reproducible
```



```

set.seed(123)
train_ind <- sample(seq_len(nrow(scaled_data)), size = smp_size)

train_sim1 <- scaled_data[train_ind,]
test_sim1 <- scaled_data[-train_ind,]

#Building Neural Network
library(h2o)
#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

## Connection successful!
##
## R is connected to the H2O cluster:
##   H2O cluster uptime:      16 minutes 41 seconds
##   H2O cluster timezone:    America/New_York
##   H2O data parsing timezone: UTC
##   H2O cluster version:     3.18.0.8
##   H2O cluster version age:  1 month and 16 days
##   H2O cluster name:        H2O_started_from_R_Shantam_Gupta_kyo754
##   H2O cluster total nodes: 1
##   H2O cluster total memory: 2.63 GB
##   H2O cluster total cores: 4
##   H2O cluster allowed cores: 4
##   H2O cluster healthy:     TRUE
##   H2O Connection ip:       localhost
##   H2O Connection port:     54321
##   H2O Connection proxy:    NA
##   H2O Internal Security:    FALSE
##   H2O API Extensions:      Algos, AutoML, Core V3, Core V4
##   R Version:                R version 3.3.3 (2017-03-06)

#import r objects to h2o cloud
train_h2o <- as.h2o(train_sim1)
test_h2o <- as.h2o(test_sim1)

set.seed(100)

dl_model_sim1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,1:4]),
  y= "RESPONSE",
  activation="Tanh",
  hidden=c(2,2),
  stopping_metric="mean_per_class_error",
  stopping_rounds = 5,
  stopping_tolerance=0.001,
  rate = 0.005, # Defaults to 0.005

```

```

mini_batch_size = 1, # defaults to 1
epochs=100,
seed = 123, # give seed
export_weights_and_biases = T, # export weights and biases defaults to false
reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
summary(dl_model_sim1)

## Model Details:
## =====
##
## H20BinomialModel: deeplearning
## Model Key: dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossE
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1      4   Input  0.00 %
## 2      2      2   Tanh  0.00 % 0.000000 0.000000  0.005736 0.005790
## 3      3      2   Tanh  0.00 % 0.000000 0.000000  0.089002 0.092362
## 4      4      2 Softmax      0.000000 0.000000  0.002051 0.000427
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000   -0.355588   0.640451 -0.017582 0.007727
## 3 0.000000    0.929416   0.993226 -0.020121 0.016930
## 4 0.000000   -2.140049   3.382085  0.000000 0.003513
##
## H20BinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.05340267
## RMSE: 0.2310902
## LogLoss: 0.2143473
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           GO NOGO      Error      Rate
## GO        74    0 0.000000   =0/74
## NOGO       0   76 0.000000   =0/76
## Totals 74   76 0.000000   =0/150
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##           metric threshold   value idx
## 1           max f1  0.555791 1.000000  75
## 2           max f2  0.555791 1.000000  75
## 3           max f0point5 0.555791 1.000000  75
## 4           max accuracy 0.555791 1.000000  75
## 5           max precision 0.982771 1.000000   0
## 6           max recall  0.555791 1.000000  75
## 7           max specificity 0.982771 1.000000   0
## 8           max absolute_mcc 0.555791 1.000000  75
## 9   max min_per_class_accuracy 0.555791 1.000000  75

```

```

## 10 max mean_per_class_accuracy 0.555791 1.000000 75
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
## H2OBinoMialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE: 0.04908478
## RMSE: 0.2215508
## LogLoss: 0.2028255
## Mean Per-Class Error: 0
## AUC: 1
## Gini: 1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      26      0 0.000000  =0/26
## NOGO      0      24 0.000000  =0/24
## Totals 26      24 0.000000  =0/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold      value idx
## 1      max f1 0.613594 1.000000 23
## 2      max f2 0.613594 1.000000 23
## 3      max f0point5 0.613594 1.000000 23
## 4      max accuracy 0.613594 1.000000 23
## 5      max precision 0.982706 1.000000 0
## 6      max recall 0.613594 1.000000 23
## 7      max specificity 0.982706 1.000000 0
## 8      max absolute_mcc 0.613594 1.000000 23
## 9      max min_per_class_accuracy 0.613594 1.000000 23
## 10 max mean_per_class_accuracy 0.613594 1.000000 23
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##      timestamp      duration training_speed      epochs iterations
## 1 2018-06-05 11:22:31 0.000 sec      0.00000      0
## 2 2018-06-05 11:22:31 0.031 sec 150000 obs/sec 1.00000      1
## 3 2018-06-05 11:22:31 0.140 sec 3191 obs/sec 2.00000      2
## 4 2018-06-05 11:22:31 0.171 sec 4128 obs/sec 3.00000      3
## 5 2018-06-05 11:22:31 0.191 sec 5504 obs/sec 4.00000      4
## 6 2018-06-05 11:22:31 0.209 sec 6302 obs/sec 5.00000      5
##      samples training_rmse training_logloss training_auc training_lift
## 1 0.000000
## 2 150.000000 0.75348 2.01605 0.28432 0.98684
## 3 300.000000 0.68028 1.39024 0.41234 0.98684
## 4 450.000000 0.53602 0.77622 0.64047 1.97368
## 5 600.000000 0.35511 0.38134 0.88549 1.97368
## 6 750.000000 0.23109 0.21435 1.00000 1.97368
##      training_classification_error validation_rmse validation_logloss
## 1
## 2 0.49333 0.75249 1.99424

```

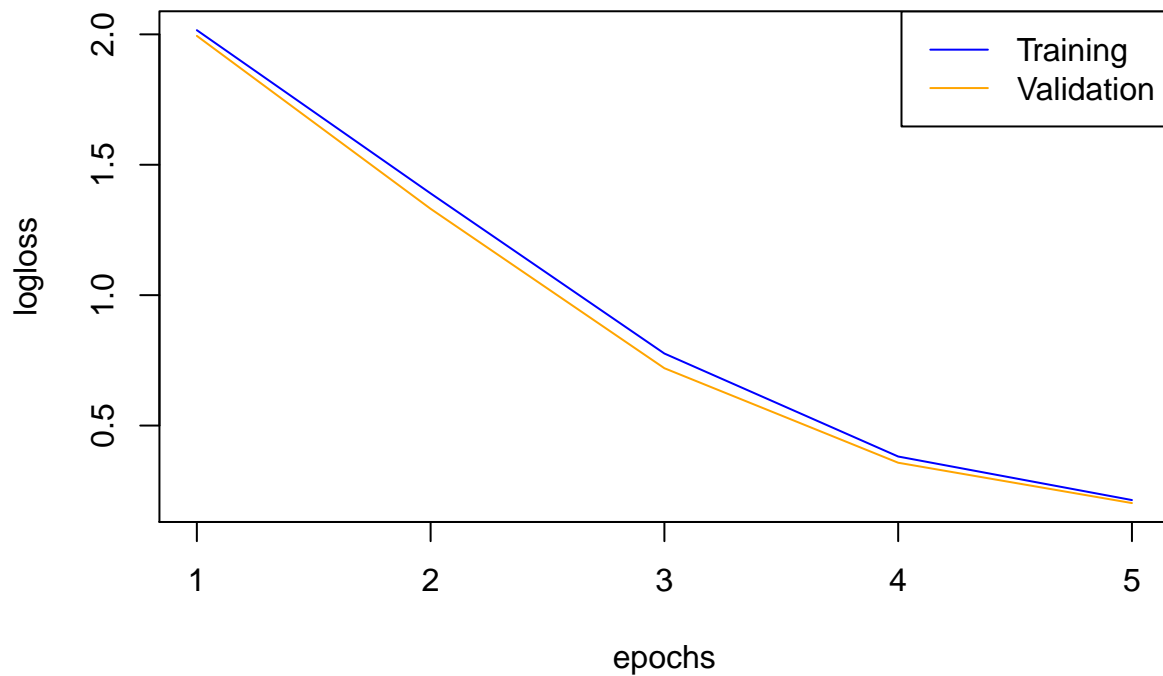
```

## 3          0.46667          0.67211          1.33114
## 4          0.33333          0.51869          0.71984
## 5          0.18000          0.34038          0.35748
## 6          0.00000          0.22155          0.20283
## validation_auc validation_lift validation_classification_error
## 1
## 2          0.27564          2.08333          0.52000
## 3          0.39103          2.08333          0.48000
## 4          0.70994          2.08333          0.40000
## 5          0.93910          2.08333          0.08000
## 6          1.00000          2.08333          0.00000
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      FWHM          1.000000          1.000000    0.386589
## 2  MassAccu          0.927834          0.927834    0.358690
## 3        RT          0.418689          0.418689    0.161861
## 4 TotalArea          0.240203          0.240203    0.092860

```

```
plot(dl_model_sim1)
```

Scoring History



8 Model with 12 Effects

- Increase in Mean (Logarithmic Drift) per feature
- Increase in 3 X Covariance per feature
- Increase in 1.5 X sigma mean shift per feature

```
#Simulation 1
#generate multivariate normal data
#parameters from a training sample
n<-1000 #incontrol observations
Data<-c()
Data0<-c()
Data1<-c()
Data2<-c()
Data3<-c()
Data4<-c()
S0<-c()

#one peptide LVNELTEFAK
mean <-c(with(data=Train,tapply(RT,INDEX=PepSeq,FUN=mean))[5],
         with(data=Train,tapply(TotalArea,INDEX=PepSeq,FUN=mean)) [5],
         with(data=Train,tapply(MassAccu,INDEX=PepSeq,FUN=mean)) [5],
         with(data=Train,tapply(FWHM,INDEX=PepSeq,FUN=mean)) [5]
)
covar<-cov(Train[Train$PepSeq=="LVNELTEFAK",c(3,6,7,8)])
#generate in-control observations

S0<-data.frame(idfile=12*n+1:(20*n),PepSeq=rep("LVNELTEFAK",n),mvrnorm(n, mean, covar))
colnames(S0)<-c("idfile","PepSeq","RT","TotalArea","MassAccu","FWHM")
#S0<- reshape(S0, idvar = "idfile", timevar = "PepSeq", direction = "wide")
RESPONSE<-c("G0")
S0 <- cbind(S0,RESPONSE)

#generate out-of-control observations
#Logarithmic drift
Data11 <-data.frame(idfile=((1):(n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(3.0*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),1.0*
                    covar))
Data12 <-data.frame(idfile=((n+1):(2*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),3.0*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),1.0*
                    covar))
Data13 <-data.frame(idfile=((2*n+1):(3*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),3.0*sqrt(covar[3,3]),1.0*
                    covar))
Data14 <-data.frame(idfile=((3*n + 1):(4*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),3.0*
                    covar))

#generate out-of-control observations for a 3 sigma fluctuation in all features large shift
covar21<- covar
covar21[1,1]<-3*covar[1,1]
```

```

Data21<-data.frame(idfile=((4*n+1):(5*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean,
                             covar21))
covar22<- covar
covar22[2,2]<-3*covar[2,2]

Data22<-data.frame(idfile=((5*n+1):(6*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean,
                             covar22))

covar23<-covar
covar23[3,3]<-3*covar[3,3]

Data23<-data.frame(idfile=((6*n+1):(7*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean,
                             covar23))

covar24<-covar
covar24[4,4]<-3*covar[4,4]

Data24<-data.frame(idfile=((7*n+1):(8*n)),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean,
                             covar24))

#generate out-of-control observations for a 1.5 sigma step shift
Data31 <-data.frame(idfile=(8*n+1):(9*n),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.5*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),1.0*sqrt(covar[4,4])),
                             covar))

Data32 <-data.frame(idfile=(9*n+1):(10*n),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),1.5*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),1.0*sqrt(covar[4,4])),
                             covar))

Data33 <-data.frame(idfile=(10*n+1):(11*n),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),1.5*sqrt(covar[3,3]),1.0*sqrt(covar[4,4])),
                             covar))

Data34 <-data.frame(idfile=(11*n+1):(12*n),PepSeq=rep("LVNELTEFAK",n),
                    mvrnorm(n, mean+c(1.0*sqrt(covar[1,1]),1.0*sqrt(covar[2,2]),1.0*sqrt(covar[3,3]),1.5*sqrt(covar[4,4])),
                             covar))

#Merge all four type of disturbances + in-control observations
Data0<-rbind(Data11,Data12, Data13, Data14, Data21, Data22, Data23, Data24, Data31, Data32, Data33, Data34)
RESPONSE<-c("NOGO")
colnames(Data0) <- c("idfile","PepSeq","RT","TotalArea","MassAccu","FWHM")
Data0 <- cbind(Data0,RESPONSE)
Data0 <-rbind(S0,Data0)

#select random ind for train and test
set.seed(123)

```

```

## 75% of the sample size
smp_size <- floor(0.8 * nrow(Data0))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(Data0)), size = smp_size)

train_sim_all <- Data0[train_ind,]
test_sim_all <- Data0[-train_ind,]

#min max scaling & centering the train data 0-1
train_maxs <- apply(train_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE)), 2, max)
train_mins <- apply(train_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE)), 2, min)
train_sim_all_scaled_data <- as.data.frame(scale(train_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE))),
                                           center = train_mins,
                                           scale = train_maxs - train_mins)

#min max scaling & centering the test data 0-1
test_maxs <- apply(test_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE)), 2, max)
test_mins <- apply(test_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE)), 2, min)
test_sim_all_scaled_data <- as.data.frame(scale(test_sim_all %>% dplyr::select(-c(idfile, PepSeq, RESPONSE))),
                                           center = test_mins,
                                           scale = test_maxs - test_mins)

train_sim_all_scaled_data$RESPONSE <- as.factor(train_sim_all$RESPONSE)
train_sim_all_scaled_data$idfile <- train_sim_all$idfile

test_sim_all_scaled_data$RESPONSE <- as.factor(test_sim_all$RESPONSE)
test_sim_all_scaled_data$idfile <- test_sim_all$idfile

#Building Neural Network
library(h2o)
#generate same set of random numbers (for reproducibility)
set.seed(121)

#launch h2o cluster
localH2O <- h2o.init(nthreads = -1)

## Connection successful!
##
## R is connected to the H2O cluster:
##   H2O cluster uptime:      16 minutes 44 seconds
##   H2O cluster timezone:    America/New_York
##   H2O data parsing timezone: UTC
##   H2O cluster version:     3.18.0.8
##   H2O cluster version age:  1 month and 16 days
##   H2O cluster name:        H2O_started_from_R_Shantam_Gupta_kyo754
##   H2O cluster total nodes: 1
##   H2O cluster total memory: 2.63 GB
##   H2O cluster total cores: 4
##   H2O cluster allowed cores: 4
##   H2O cluster healthy:     TRUE
##   H2O Connection ip:       localhost
##   H2O Connection port:     54321
##   H2O Connection proxy:    NA
##   H2O Internal Security:    FALSE

```

```

##      H2O API Extensions:      Algos, AutoML, Core V3, Core V4
##      R Version:              R version 3.3.3 (2017-03-06)

#import r objects to h2o cloud
train_h2o <- as.h2o(train_sim_all)
test_h2o <- as.h2o(test_sim_all)

set.seed(100)

dl_model_sim1 <- h2o.deeplearning(
  model_id="dl_model_first",
  training_frame=train_h2o,
  validation_frame = test_h2o,
  x= colnames(train_h2o[,3:6]),
  y= "RESPONSE",
  activation="Tanh",
  hidden=c(20,20),
  standardize = TRUE, #standardizes the data
  loss= "CrossEntropy",
  stopping_metric="logloss",
  stopping_rounds = 10,
  stopping_tolerance=0.00001,
  adaptive_rate = TRUE,
  shuffle_training_data = TRUE,
  rate = 0.005, # Defaults to 0.005 adaptive enabled so cannot specify the learning rate
  mini_batch_size = 1, # defaults to 1
  epochs=200,
  seed = 123, # give seed
  export_weights_and_biases = T, # export weights and biases defaults to false
  reproducible = T # Force reproducibility on small data (will be slow - only uses 1 thread). Defaults
)
summary(dl_model_sim1)

## Model Details:
## =====
##
## H2OBinomialModel: deeplearning
## Model Key:  dl_model_first
## Status of Neuron Layers: predicting RESPONSE, 2-class classification, bernoulli distribution, CrossEntropy
batch size 1
##   layer units   type dropout      l1      l2 mean_rate rate_rms
## 1      1      4   Input  0.00 %
## 2      2     20   Tanh  0.00 % 0.000000 0.000000  0.001635 0.001284
## 3      3     20   Tanh  0.00 % 0.000000 0.000000  0.104984 0.104947
## 4      4      2 Softmax      0.000000 0.000000  0.004494 0.001813
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000    0.008561   0.537289 -0.170672 0.505694
## 3 0.000000    0.002239   0.563893 -0.194312 0.880953
## 4 0.000000    0.037249   0.973738 -0.000000 2.371749
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 10003 samples **
##

```



```

## MSE: 0.1200252
## RMSE: 0.3464466
## LogLoss: 0.3870544
## Mean Per-Class Error: 0.1837601
## AUC: 0.863129
## Gini: 0.7262579
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      5780  476 0.076087   =476/6256
## NOGO     1092 2655 0.291433   =1092/3747
## Totals 6872 3131 0.156753   =1568/10003
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold   value idx
## 1      max f1  0.336488 0.772027 214
## 2      max f2  0.150029 0.775119 320
## 3      max f0point5 0.632382 0.848559 141
## 4      max accuracy 0.507659 0.849145 168
## 5      max precision 0.998129 1.000000 0
## 6      max recall  0.067311 1.000000 399
## 7      max specificity 0.998129 1.000000 0
## 8      max absolute_mcc 0.507659 0.678064 168
## 9      max min_per_class_accuracy 0.214707 0.785962 268
## 10     max mean_per_class_accuracy 0.336488 0.816240 214
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/I>)`
## H2OBinoMialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## MSE: 0.1210775
## RMSE: 0.3479619
## LogLoss: 0.3900262
## Mean Per-Class Error: 0.1817058
## AUC: 0.8617871
## Gini: 0.7235742
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##      GO NOGO      Error      Rate
## GO      3785  205 0.051378   =205/3990
## NOGO     752 1658 0.312033   =752/2410
## Totals 4537 1863 0.149531   =957/6400
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##      metric threshold   value idx
## 1      max f1  0.403103 0.776036 188
## 2      max f2  0.150023 0.774529 323
## 3      max f0point5 0.610394 0.846334 140
## 4      max accuracy 0.414126 0.850625 186
## 5      max precision 0.998243 1.000000 0
## 6      max recall  0.069866 1.000000 398
## 7      max specificity 0.998243 1.000000 0
## 8      max absolute_mcc 0.414126 0.679527 186

```

```

## 9  max min_per_class_accuracy  0.216114 0.785062 270
## 10 max mean_per_class_accuracy  0.403103 0.818294 188
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##      timestamp  duration training_speed  epochs iterations
## 1 2018-06-05 11:22:35  0.000 sec           0.00000      0
## 2 2018-06-05 11:22:36  0.794 sec   39813 obs/sec 1.00000      1
## 3 2018-06-05 11:22:37  1.580 sec   39475 obs/sec 2.00000      2
## 4 2018-06-05 11:22:37  2.365 sec   39384 obs/sec 3.00000      3
## 5 2018-06-05 11:22:38  3.150 sec   39430 obs/sec 4.00000      4
##      samples training_rmse training_logloss training_auc training_lift
## 1      0.000000
## 2 25600.000000      0.36975      0.43144      0.84593      2.66960
## 3 51200.000000      0.35592      0.40547      0.85339      2.66960
## 4 76800.000000      0.35602      0.40475      0.85625      2.66960
## 5 102400.000000      0.35727      0.40764      0.85561      2.66960
##      training_classification_error validation_rmse validation_logloss
## 1
## 2      0.16465      0.37272      0.43810
## 3      0.16645      0.35816      0.41039
## 4      0.16355      0.35821      0.41112
## 5      0.16085      0.35890      0.41150
##      validation_auc validation_lift validation_classification_error
## 1
## 2      0.84254      2.65560      0.16672
## 3      0.85139      2.65560      0.16531
## 4      0.85166      2.65560      0.16391
## 5      0.85475      2.65560      0.16078
##
## ---
##      timestamp  duration training_speed  epochs iterations
## 60 2018-06-05 11:23:18 42.962 sec  43174 obs/sec 59.00000      59
## 61 2018-06-05 11:23:19 43.660 sec  43198 obs/sec 60.00000      60
## 62 2018-06-05 11:23:19 44.374 sec  43210 obs/sec 61.00000      61
## 63 2018-06-05 11:23:20 45.075 sec  43256 obs/sec 62.00000      62
## 64 2018-06-05 11:23:21 45.777 sec  43281 obs/sec 63.00000      63
## 65 2018-06-05 11:23:21 45.932 sec  43281 obs/sec 63.00000      63
##      samples training_rmse training_logloss training_auc
## 60 1510400.000000      0.35224      0.40197      0.86271
## 61 1536000.000000      0.34926      0.39379      0.86059
## 62 1561600.000000      0.34745      0.38957      0.86076
## 63 1587200.000000      0.34645      0.38705      0.86313
## 64 1612800.000000      0.34909      0.39267      0.86366
## 65 1612800.000000      0.34645      0.38705      0.86313
##      training_lift training_classification_error validation_rmse
## 60      2.66960      0.15255      0.35452
## 61      2.66960      0.15555      0.35167
## 62      2.66960      0.14956      0.34926
## 63      2.66960      0.15675      0.34796
## 64      2.66960      0.15075      0.35193
## 65      2.66960      0.15675      0.34796

```

```
## validation_logloss validation_auc validation_lift
## 60 0.40690 0.86043 2.65560
## 61 0.39828 0.85978 2.65560
## 62 0.39369 0.85832 2.65560
## 63 0.39003 0.86179 2.65560
## 64 0.39799 0.85894 2.65560
## 65 0.39003 0.86179 2.65560
## validation_classification_error
## 60 0.15375
## 61 0.15438
## 62 0.15156
## 63 0.14953
## 64 0.15047
## 65 0.14953
##
## Variable Importances: (Extract with `h2o.varimp`)
## =====
##
## Variable Importances:
## variable relative_importance scaled_importance percentage
## 1 RT 1.000000 1.000000 0.277106
## 2 TotalArea 0.933698 0.933698 0.258734
## 3 FWHM 0.906112 0.906112 0.251089
## 4 MassAccu 0.768915 0.768915 0.213071
```

```
plot(dl_model_sim1)
```

Scoring History

