Particle Identification ML

Gerhard Viljoen

12/22/2018

require(tufte)

### Set seed for reproducibility:

set.seed(123456789)

# Read in all the json files, created from python dictionaries

rm(list=ls())  
  
require(jsonlite)  
require(readtext)  
  
#PDG codes  
pdg.elec <- c(11,-11)  
pdg.pion <- c(-211,211)  
  
files <- list.files(path="~/Thesis data/SemiFullData", pattern="\*json", full.names=T, recursive=FALSE)  
  
j <- fromJSON(files[1])  
  
for(i in 2:length(files)){  
  
 f <- fromJSON(files[i])  
 j <- c(j,f)  
}  
  
length(j)  
  
save(j,file="~/Thesis data/SemiFullData/fulljson.rdata")

# Threshold pad data

Load full json record of all extracted events:

rm(list=ls())  
load("~/Thesis data/SemiFullData/fulljson.rdata")

Extract pad data

pads <- sapply(j,`[[`,"layer0")

Replace all pads that have data, with only those bin-rows which are all non-zero

no.zero.row.pads <- list()  
  
for(i in 1:length(pads)){  
 if(is.null(pads[[i]])){  
 no.zero.row.pads[[i]] <- NULL  
 }else if(typeof(pads[[i]])=="list"){  
 no.zero.row.pads[[i]] <- NULL  
 }else{  
 this.pad <- as.matrix(pads[[i]])  
 r <- rowSums(this.pad)  
 r <- which(r==0)  
  
 if(length(r)==11){  
 this.pad <- NULL  
 }else{  
 this.pad <- this.pad[-r,]  
 }  
  
  
  
 no.zero.row.pads[[i]] <- this.pad  
 }  
}  
  
save(no.zero.row.pads,file="~/Thesis data/SemiFullData/no\_zero\_row\_pads.rdata")

Create look-up tables to find pad data for both electrons and pions, and add an indicator column for which entries have no pad data.

Also create an indicator of which tracks’ trackIDs are 0, since this may indicate that an error has occurred

tracks <- sapply(j, `[[`,"track")  
tracks <- as.numeric(tracks)  
  
  
pdg <- sapply(j, `[[`,"pdgCode")  
pdg <- as.numeric(pdg)  
  
exclude <- rep(0,length(no.zero.row.pads))  
  
for(i in 1:length(no.zero.row.pads)){  
 if(is.null(no.zero.row.pads[[i]])){  
 exclude[i] <- 1  
 }  
}  
  
exclude2 <- rep(0,length(no.zero.row.pads))  
  
for(i in 1:length(tracks)){  
 if(tracks[i]==0){  
 exclude2[i] <- 1  
 }  
}  
  
  
look.up.table <- data.frame(cbind(pdg,1:length(pdg),exclude,exclude2))  
names(look.up.table) <- c("pdg","index","exclude.null.pads","exclude.track.id.0")

require(dplyr)  
pdg.elec <- c(11,-11)  
pdg.pion <- c(211,-211)  
electrons <- look.up.table[which(look.up.table$pdg %in% pdg.elec),]  
pions <- look.up.table[which(look.up.table$pdg %in% pdg.pion),]  
  
electrons <- unique(electrons)  
pions <- unique(pions)  
  
electrons <- electrons %>%  
 subset(exclude.null.pads==0) %>%  
 subset(exclude.track.id.0==0)  
  
pions <- pions %>%  
 subset(exclude.null.pads==0) %>%  
 subset(exclude.track.id.0==0)  
  
electrons <- as.data.frame(electrons)  
pions <- as.data.frame(pions)  
  
electrons <- electrons$index  
pions <- pions$index  
  
save(electrons,file="~/Thesis data/SemiFullData/electrons.rdata")  
save(pions,file="~/Thesis data/SemiFullData/pions.rdata")  
save(pads,file="~/Thesis data/SemiFullData/pads.rdata")

# Descriptive statistics

Get total energy deposition per pad, for pions and electrons, respectively:

electron.pad.sum <- c()  
  
for(i in electrons){  
 this.pad.sum <- sum(as.numeric(pads[[i]]))  
 electron.pad.sum <- c(electron.pad.sum,this.pad.sum)  
}  
  
pion.pad.sum <- c()  
  
for(i in pions){  
 this.pad.sum <- sum(as.numeric(pads[[i]]))  
 pion.pad.sum <- c(pion.pad.sum,this.pad.sum)  
}

electrons <- data.frame(cbind(electrons,electron.pad.sum))  
pions <- data.frame(cbind(pions,pion.pad.sum))  
  
names(electrons) <- c("index","total.charge.deposit")  
names(pions) <- c("index","total.charge.deposit")

Get 5 number summary statistics, for each pad where 0-sum bin-rows have been removed.

electron.pad.summary <- numeric(6)  
  
for(i in electrons$index){  
 this.pad <- summary(as.numeric(no.zero.row.pads[[i]]))  
 electron.pad.summary <- rbind(electron.pad.summary,this.pad)  
}  
  
electron.pad.summary <- as.data.frame(electron.pad.summary)  
names(electron.pad.summary) <- c("Min","1Q","Median","Mean","3Q","Max")  
  
pion.pad.summary <- numeric(6)  
  
for(i in pions$index){  
 this.pad <- summary(as.numeric(no.zero.row.pads[[i]]))  
 pion.pad.summary <- rbind(pion.pad.summary,this.pad)  
}  
  
pion.pad.summary <- as.data.frame(pion.pad.summary)  
names(pion.pad.summary) <- c("Min","1Q","Median","Mean","3Q","Max")

Clean up:

electron.pad.summary <- electron.pad.summary[-1,]  
pion.pad.summary <- pion.pad.summary[-1,]  
  
electrons <- data.frame(cbind(electrons,electron.pad.summary))  
pions <- data.frame(cbind(pions,pion.pad.summary))  
  
  
save(electrons,file="~/Thesis data/SemiFullData/electrons.rdata")  
save(pions,file="~/Thesis data/SemiFullData/pions.rdata")

Get the number of non-zero bin-rows per detector pad, for electrons and pions:

electron.timebins <- c()  
  
for(i in electrons$index){  
  
 if(is.null(nrow(no.zero.row.pads[[i]]))){  
 this.num.bin <- 0  
 }else{  
 this.num.bin <- nrow(no.zero.row.pads[[i]])  
 }  
  
  
 electron.timebins <- c(electron.timebins,this.num.bin)  
  
}  
  
pion.timebins <- c()  
  
for(i in pions$index){  
  
 if(is.null(nrow(no.zero.row.pads[[i]]))){  
 this.num.bin <- 0  
 }else{  
 this.num.bin <- nrow(no.zero.row.pads[[i]])  
  
  
 }  
  
 pion.timebins <- c(pion.timebins,this.num.bin)  
}  
  
  
electrons$num.bins <- electron.timebins  
pions$num.bins <- pion.timebins  
  
save(electrons,file="~/Thesis data/SemiFullData/electrons.rdata")  
save(pions,file="~/Thesis data/SemiFullData/pions.rdata")

Get the column means for each pad, where non-zero time-bins have been removed:

electron.compressed.bins <- numeric(24)  
  
for(i in electrons$index){  
  
 if(is.null(dim(no.zero.row.pads[[i]]))){  
 this.entry <- as.numeric(no.zero.row.pads[[i]])  
 }else{  
  
this.entry <- as.numeric(colSums(no.zero.row.pads[[i]]))  
  
  
 }  
  
 electron.compressed.bins <- rbind(electron.compressed.bins,this.entry)  
}  
  
pion.compressed.bins <- c()  
  
for(i in pions$index){  
  
 if(is.null(dim(no.zero.row.pads[[i]]))){  
 this.entry <- as.numeric(no.zero.row.pads[[i]])  
 }else{  
  
 this.entry <- as.numeric(colSums(no.zero.row.pads[[i]]))  
  
 }  
 pion.compressed.bins <- rbind(pion.compressed.bins,this.entry)  
}  
  
electron.compressed.bins <- electron.compressed.bins[-1,]  
  
pion.compressed.bins <- pion.compressed.bins[-1,]  
  
electron.compressed.bins <- as.data.frame(electron.compressed.bins)  
pion.compressed.bins <- as.data.frame(pion.compressed.bins)  
  
names <- paste0("c",1:24)  
  
names(electron.compressed.bins) <- names  
names(pion.compressed.bins) <- names  
  
electrons <- data.frame(cbind(electrons,electron.compressed.bins))  
pions <- data.frame(cbind(pions,pion.compressed.bins))  
  
save(electrons,file="~/Thesis data/SemiFullData/electrons.rdata")  
save(pions,file="~/Thesis data/SemiFullData/pions.rdata")  
  
electrons$id <- "electron"  
pions$id <- "pion"  
  
electrons$id <- as.factor(electrons$id)  
pions$id <- as.factor(pions$id)

dat <- data.frame(rbind(electrons,pions))  
save(dat,file="~/Thesis data/SemiFullData/unscaled\_data.rdata")

Scale data

dat[,2:33] <- scale(dat[,2:33])  
save(dat,file="~/Thesis data/SemiFullData/scaled\_data.rdata")  
  
load("~/Thesis data/SemiFullData/scaled\_data.rdata")  
i <- dat$index  
rm(dat)

Remove missing values

load("~/Thesis data/SemiFullData/scaled\_data.rdata")  
dat <- na.omit(dat)

rownames(dat) <- NULL  
  
index <- dat$index  
  
dat <- dat[,-1]  
  
dat$id <- as.character(dat$id)  
  
dat$electron <- ifelse(dat$id=="electron",1,0)  
  
id <- dat$id  
  
dat <- dat[,-1]

Get a equal sample of electrons and pions (50 000 each):

id.index <- data.frame(cbind(id,index))  
  
require(dplyr)  
  
electron.id <- id.index[id.index$id=="electron",]  
  
pion.id <- id.index[id.index$id=="pion",]  
  
electron.train <- base::sample(electron.id$index,50000,replace = F)  
pion.train <- base::sample(pion.id$index, 50000,replace=F)  
  
electron.train <- as.numeric(as.character(electron.train))  
pion.train <- as.numeric(as.character(pion.train))

Since particles pass through pads at different angles, and at different coordinates within a pad, the exact column/ row in the matrix representation of a detector trace is not important.

To this end, the actual pad data (“images”) will not be included in the feature set used for vanilla feed-forward neural networks, to get an early estimation of particle ID based on summary statistics.

At a later stage, kernels/ masks will be employed in a convolutional neural network setup, to detect features in the detector “images” that are diagnostic of the “electron”/ “pion” ID.

Furthermore, the 24 column sums can only be informative insomuch as they are averaged across all columns, since the positional information is not relevant to the PID process:

dat <- data.frame(cbind(index,dat))  
  
c <- dat[,9:32]  
  
c <- rowMeans(c)  
  
dat <- dat[,c(1:8,33,34)]  
  
dat <- data.frame(cbind(c,dat))  
  
dat <- dat[,-10]  
  
save(dat,file="~/Thesis data/SemiFullData/final\_data.rdata")  
rm(c)  
  
train.id <- c(electron.train,pion.train)  
  
train <- dat %>%  
 subset(index %in% train.id)  
  
test <- dat %>%  
 subset(! index %in% train.id)  
  
save(train,file="~/Thesis data/SemiFullData/train1.rdata")  
save(test,file="~/Thesis data/SemiFullData/test1.rdata")

load("~/Thesis data/SemiFullData/train1.rdata")  
load("~/Thesis data/SemiFullData/test1.rdata")

Separate out index from data:

test.id.index <- test[,c(2,10)]  
train.id.index <- train[,c(2,10)]  
  
test <- test[,-2]  
train <- train[,-2]

# Particle Identification:

## Majority Class Classifier:

The vast majority of particles detected in the TRD are pions, in fact in our dataset we have the following ratio:

PID <- c(train$electron,test$electron)  
  
pion.percentage <- 100-sum(PID/length(PID))  
electron.percentage <- sum(PID/length(PID))  
  
t <- data.frame(pion.percentage,electron.percentage)  
require(knitr)  
kable(t)

|  |  |
| --- | --- |
| pion.percentage | electron.percentage |
| 99.89238 | 0.1076221 |

It is clear, then, that our task is not smiply as classification problem, since predicting “pion” for each particle will result in upwards of 99% accuracy, without any effort.

Our task centers around pion efficiency, i.e. not misclassifying a pion as an electron, while not losing too many electrons in the process.

## Confusion matrix:

PID <- data.frame(cbind(PID,rep(0,length(PID))))  
names(PID) <- c("GroundTruth","MajorityClassClassifierPred")  
  
PID$GroundTruth <- as.factor(PID$GroundTruth)  
PID$MajorityClassClassifierPred <- as.factor(PID$MajorityClassClassifierPred)  
  
levels(PID$MajorityClassClassifierPred) <- c("0","1")  
  
kable(table(PID))

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 539247 | 0 |
| 1 | 65034 | 0 |

negs <- c("True Negative","False Negative")  
poss <- c("False Positives","True Positives")  
  
conf.m <- data.frame(cbind(negs,poss))  
names(conf.m) <- c("Predicted PION","Predicted ELECTRON")  
rownames(conf.m) <- c("IS a PION", "IS an ELECTRON")  
  
kable(conf.m)

|  |  |  |
| --- | --- | --- |
|  | Predicted PION | Predicted ELECTRON |
| IS a PION | True Negative | False Positives |
| IS an ELECTRON | False Negative | True Positives |

A more realistic evaluation of our Majority Classifier model accuracy, would be as follows:

We reject 100% of Pions, but we keep 0% of electrons.

Our goal is to find a model that rejects as many pions as possible, but keeps as many electrons as possible as well.

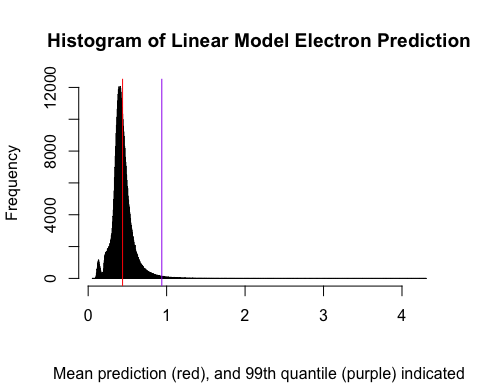
## Simple linear regression:

linmod1 <- lm(electron~.,data=train)  
  
linmod.preds <- predict(linmod1,test)

We plot the distribution of electron predictions, where electrons were labeled as integer 1 and pions as integer 0, and add reference lines for the mean prediction in red and the 99th quantile in purple.

If we go with our prior knowledge that around 99.9% of particles detected in the TRD, the 99th quantile seems like a reasonable cut-off point at this point to get a benchmark for pion rejection.

accuracy <- data.frame(cbind(as.numeric(test$electron),linmod.preds))  
  
names(accuracy) <- c("actual","Linear Model Prediction")  
  
r <- range(accuracy$`Linear Model Prediction`)  
  
q <- quantile(accuracy$`Linear Model Prediction`,0.99)  
  
hist(accuracy$`Linear Model Prediction`,breaks=1000,main="Histogram of Linear Model Electron Prediction",sub="Mean prediction (red), and 99th quantile (purple) indicated",xlab="")  
abline(v=mean(accuracy$`Linear Model Prediction`),col="red")  
abline(v=q,col="purple")



We apply this logic to get a binary response value from our basic linear model, and show the confusion matrix for this prediction:

accuracy$lm1.99quantile.pred <- ifelse(accuracy$`Linear Model Prediction`>=q,1,0)  
  
accuracy$actual <- as.factor(accuracy$actual)  
  
accuracy$lm1.99quantile.pred <- as.factor(accuracy$lm1.99quantile.pred)  
  
kable(table(accuracy$actual,accuracy$lm1.99quantile.pred))

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 484793 | 4454 |
| 1 | 14445 | 589 |

Assessing our model accuracy:

accuracy$actual <- as.numeric(as.character(accuracy$actual))  
accuracy$lm1.99quantile.pred <- as.numeric(as.character(accuracy$lm1.99quantile.pred))  
accuracy$correct <- ifelse(accuracy$actual==accuracy$lm1.99quantile.pred,1,0)  
  
a <- sum(accuracy$correct/nrow(accuracy))

## [1] "We achieve an accuracy score of 0.962522879109071"

### Pion Efficiency:

pion.efficiency <- ifelse(accuracy$actual==0&accuracy$lm1.99quantile.pred==0,1,0)  
p <- length(which(accuracy$actual==0)==T)  
pion.efficiency <- sum(pion.efficiency)/p  
  
pion.efficiency <- 100-pion.efficiency\*100

## [1] "Our majority class classifier incorrectly classified 0.10762% of pions as electrons"

## [1] "Based on predictions from our basic linear model, we incorrectly classify 0.91038% of pions in our test set as electrons"

### Electron Efficiency:

electron.efficiency <- ifelse(accuracy$actual==1&accuracy$lm1.99quantile.pred==1,1,0)  
e <- length(which(accuracy$actual==1)==T)  
electron.efficiency <- sum(electron.efficiency)/e  
  
electron.efficiency <- electron.efficiency\*100

## [1] "Our majority class classifier correctly accepted 0% of electrons in our test set"

## [1] "Based on predictions from our basic linear model, we correctly accept 3.91779% of electrons in our test set"

## Optimization of linear model prediction cut-off point for electron classification:

library(pROC)  
  
#Penalize pion error and electron error equally:  
  
lm.optimization <- function(cut.off){  
 p <- predict(linmod1,test)  
 a <- test$electron  
   
 my.prediction <- ifelse(p>=cut.off,1,0)  
   
 roc\_obj <- roc(a, my.prediction)  
 auc(roc\_obj)  
   
}  
  
optim.q <- optim(par=r[1],fn=lm.optimization,lower=r[1],upper=r[2],method="Brent",control = list(fnscale=-1))  
  
q <- optim.q$par

## [1] "After optimizing the electron prediction cut-off, by maximizing the area under the curve of the receiver operating characteristic, an AUC value of 0.709412331525969 is achieved"

Our new confusion matrix, with a more informed cut-off point, looks as follows:

accuracy$lm1.99quantile.pred <- ifelse(accuracy$`Linear Model Prediction`>=q,1,0)  
  
accuracy$actual <- as.factor(accuracy$actual)  
  
accuracy$lm1.99quantile.pred <- as.factor(accuracy$lm1.99quantile.pred)  
  
kable(table(accuracy$actual,accuracy$lm1.99quantile.pred))

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 367069 | 122178 |
| 1 | 4983 | 10051 |

### Optimized Cut-off linear model accuracy assessment:

accuracy$actual <- as.numeric(as.character(accuracy$actual))  
accuracy$lm1.99quantile.pred <- as.numeric(as.character(accuracy$lm1.99quantile.pred))  
accuracy$correct <- ifelse(accuracy$actual==accuracy$lm1.99quantile.pred,1,0)  
  
a <- sum(accuracy$correct/nrow(accuracy))

## [1] "We achieve an accuracy score of 0.74783701943956"

### Pion Efficiency:

old.pion.efficiency <- pion.efficiency  
  
pion.efficiency <- ifelse(accuracy$actual==0&accuracy$lm1.99quantile.pred==0,1,0)  
p <- length(which(accuracy$actual==0)==T)  
pion.efficiency <- sum(pion.efficiency)/p  
  
pion.efficiency <- 100-pion.efficiency\*100

## [1] "Our majority class classifier incorrectly classified 0.10762% of pions as electrons"

## [1] "Based on predictions from our basic linear model, we incorrectly classify 0.91038% of pions in our test set as electrons"

## [1] "Based on predictions from our cut-off optimized basic linear model, we incorrectly classify 24.97266% of pions in our test set as electrons"

### Electron Efficiency:

old.electron.efficiency <- electron.efficiency  
  
electron.efficiency <- ifelse(accuracy$actual==1&accuracy$lm1.99quantile.pred==1,1,0)  
e <- length(which(accuracy$actual==1)==T)  
electron.efficiency <- sum(electron.efficiency)/e  
  
electron.efficiency <- electron.efficiency\*100

## [1] "Our majority class classifier correctly accepted 0% of electrons in our test set"

## [1] "Based on predictions from our basic linear model, we correctly accept 3.91779% of electrons in our test set"

## [1] "Based on predictions from our cut-off optimized basic linear model, we correctly accept 66.85513% of electrons in our test set as electrons"

# DataRobot: Automated ML

DataRobot is a platform for automated ML, which automates data-preprocessing, feature generation, model building with cross validation, and prediction API deployment.

In order to find models which could be useful in PID, the full training and test set were uploaded onto their platform and “Autopilot” was initialized.

Figure one shows the feature importance results after data upload:

datarobot.train <- rbind(train,test)  
datarobot.train$electron <- ifelse(datarobot.train$electron==1,"electron","pion")  
write.csv(datarobot.train,"~/MSc-train-data.csv")

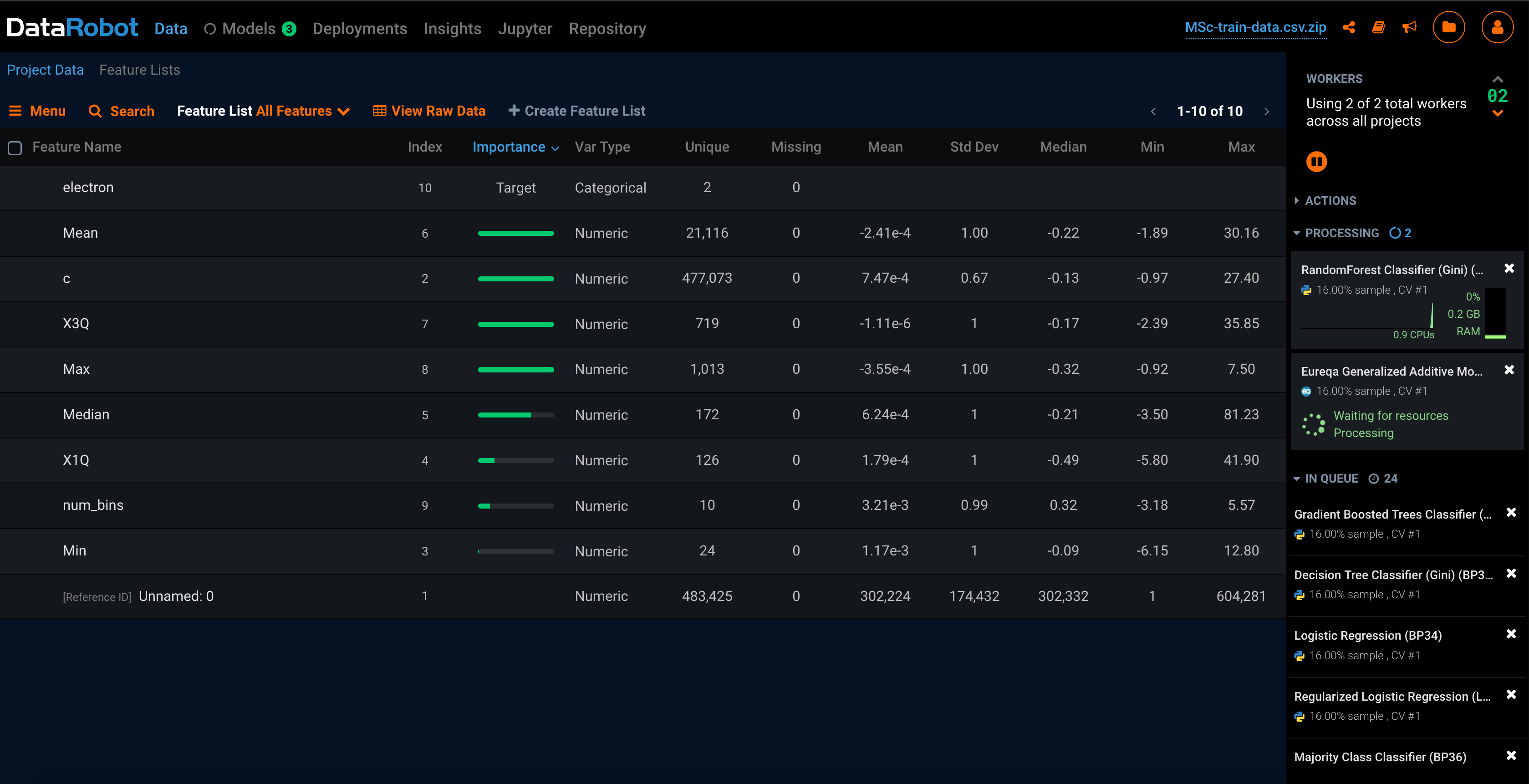


Figure1: Feature Importance

After running models overnight, DataRobot built 58 models, of which the top 5 are shown here:

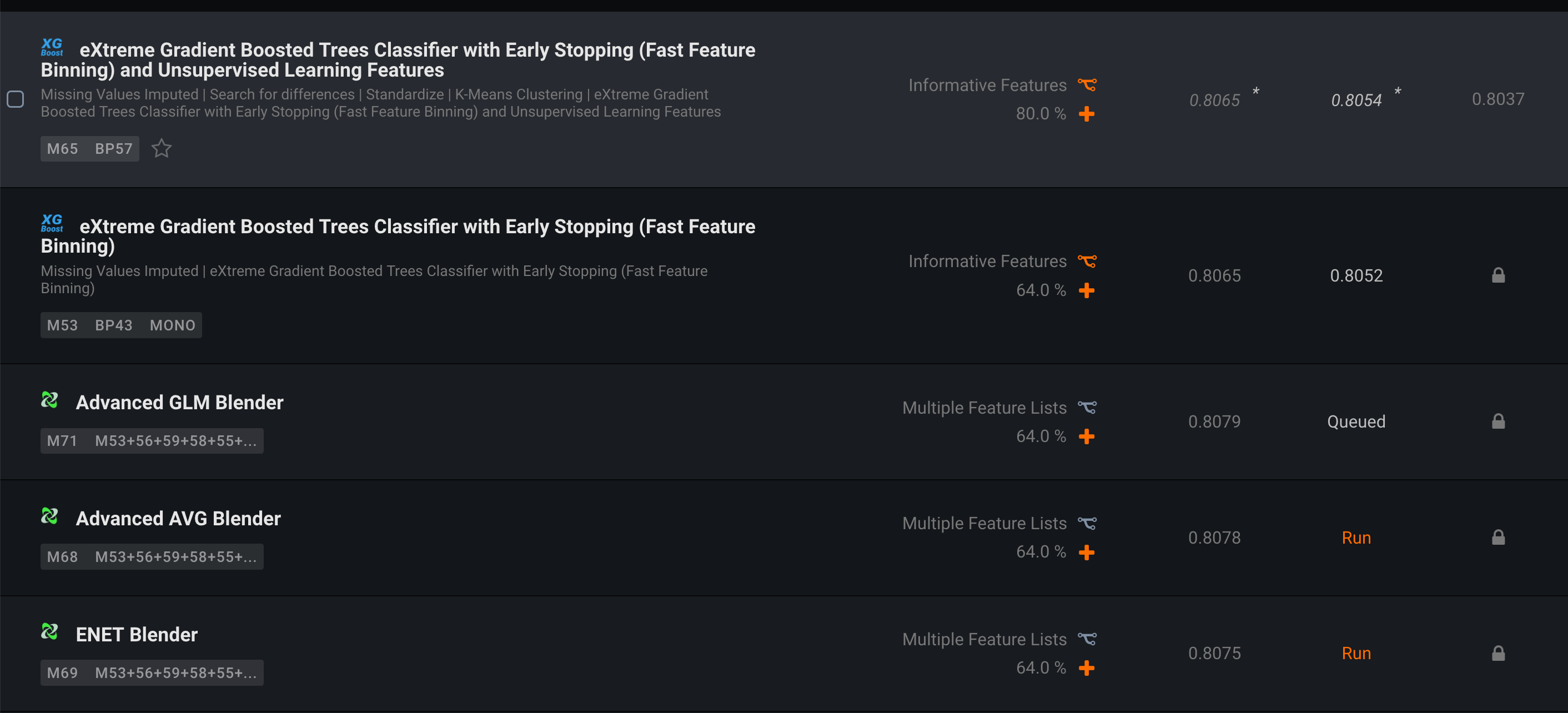


Figure 2: Top 5 DataRobot models

Evaluating the top model’s metrics, shows the following model blueprint:

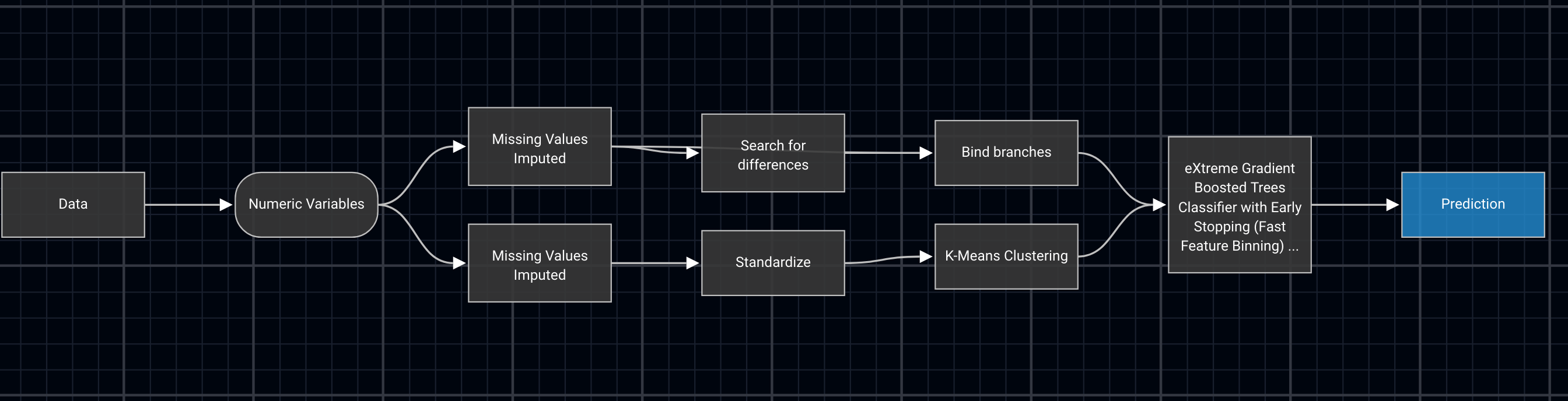


Figure 3: Extreme Gradient Boosted Tree Classifier with Early Stopping (Fast Feature Binning) Bueprint

The ROC curve and Prediction Distribtution plots are as follows:

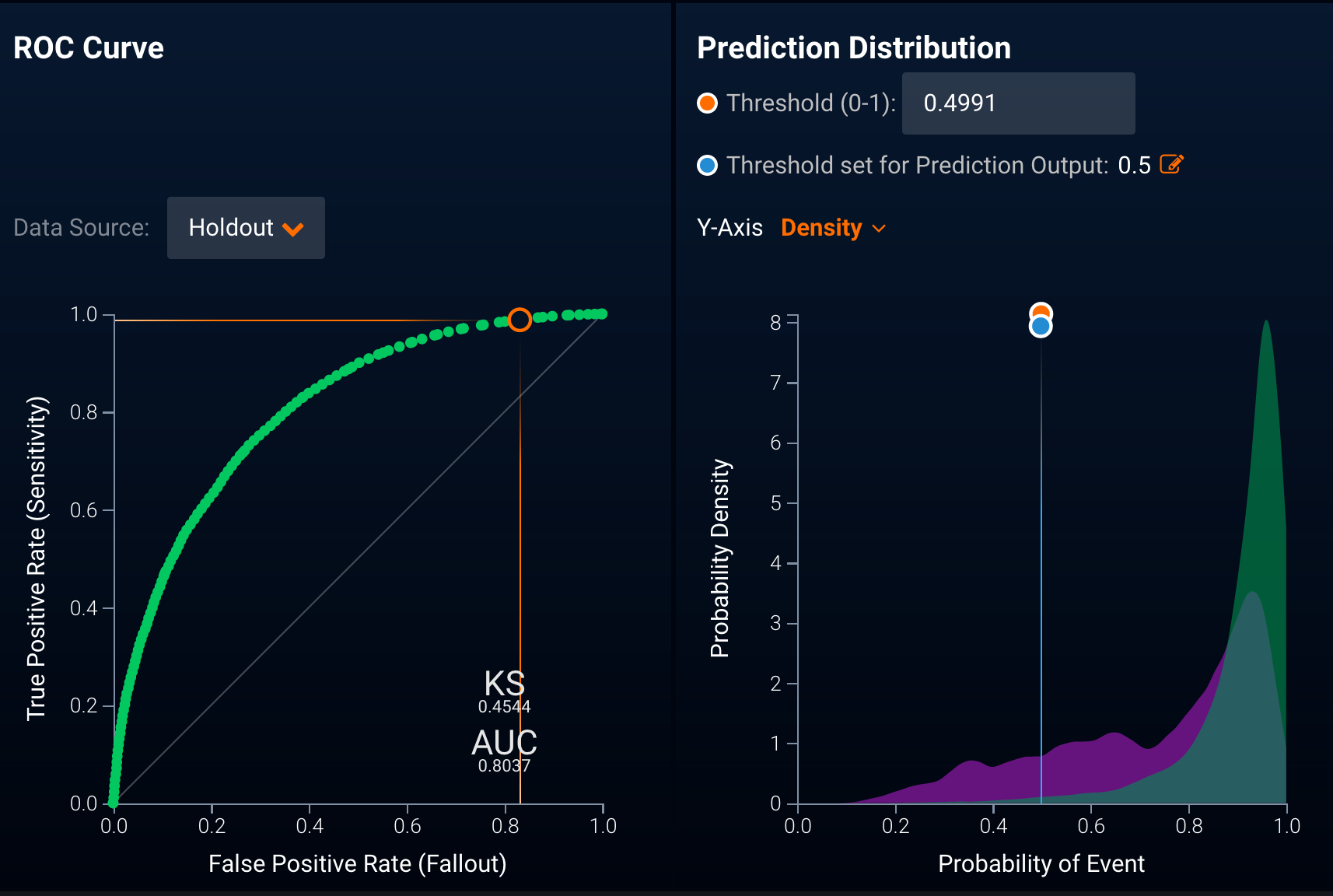


Figure 4: Top Model Evaluation

DataRobot’s Confusion Matrix is flipped relative to what has been shown so far, i.e. IS(pion) +/ IS NOT(pion) -



Figure 5: Top Model Confusion Matrix

## [1] "DataRobot detects 16.9139694010917% of electrons in the holdout set it created from the FULL dataset that was uploaded"

## [1] "DataRobot incorrectly classifies 1.14697400995837% of pions as electrons in the holdout set it created from the FULL dataset that was uploaded"

# Deep Learning

rm(list=ls())  
  
load("~/Thesis data/SemiFullData/fulljson.rdata")  
  
pads <- sapply(j,`[[`,"layer0")  
rm(j)

## Roll out pad data

require(Smisc)  
  
pad.df <- sapply(pads,as.numeric)  
rm(pads)  
  
for(i in 1:length(pad.df)){  
 if(length(pad.df[[i]])!=264){  
 pad.df[[i]] <- as.numeric(rep(0,264))  
 }  
}  
  
pad.df <- do.call(rbind,pad.df)  
  
save(pad.df,file="~/Thesis data/SemiFullData/cleanpads.rdata")

load("~/Thesis data/SemiFullData/fulljson.rdata")  
load("~/Thesis data/SemiFullData/cleanpads.rdata")  
pdg <- sapply(j,`[[`,"pdgCode")  
  
pdg <- as.numeric(pdg)  
  
pdg <- as.data.frame(pdg)  
pdg$index <- rownames(pdg)  
  
pdg$pdg <- ifelse(pdg$pdg %in% c(11,-11),T,F)  
  
names(pdg) <- c("electron","index")  
  
pdg <- pdg[,c(2,1)]  
  
cnn.dat <- data.frame(cbind(pdg,pad.df))  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn.rdata")

load("~/Thesis data/SemiFullData/cnn.rdata")  
  
cnn.dat <- as.matrix(cnn.dat[,-c(1,2)])  
  
zero.pads <- c()  
  
for(i in 1:nrow(cnn.dat)){  
 if(all(cnn.dat[i,]==0)){  
 zero.pads <- c(zero.pads,i)  
 }  
}  
  
cnn.dat <- cnn.dat[-zero.pads,]  
pdg <- pdg[-zero.pads,]  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn.rdata")  
save(pdg,file="~/Thesis data/SemiFullData/pdg.rdata")

set.seed(123456)  
pdg$index <- 1:nrow(pdg)  
  
require(dplyr)  
  
e.ind <- pdg %>%  
 subset(electron==T)  
  
p.ind <- pdg %>%  
 subset(electron==F)  
  
e.holdout <- base::sample(e.ind$index,size=5000,replace=F)  
p.holdout <- base::sample(p.ind$index,size=5000,replace=F)  
  
e.ind <- e.ind %>%  
 subset(!index %in% e.holdout)  
  
p.ind <- p.ind %>%  
 subset(!index %in% p.holdout)  
  
holdout.ind <- as.numeric(c(e.holdout,p.holdout))  
  
holdout.data <- cnn.dat[holdout.ind,]  
  
cnn.dat <- cnn.dat[-holdout.ind,]  
  
pdg.holdout <- pdg[holdout.ind,]  
  
pdg <- pdg[-holdout.ind,]  
  
pdg$index <- 1:nrow(pdg)  
pdg.holdout$index <- 1:nrow(pdg.holdout)  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn.rdata")  
save(holdout.data,file="~/Thesis data/SemiFullData/holdout.rdata")  
save(pdg,file="~/Thesis data/SemiFullData/pdg.rdata")  
save(pdg.holdout,file="~/Thesis data/SemiFullData/pdg.holdout.rdata")

## Unbalanced class compensation:

Resample electrons 10 times:

e.bootstrap.ind <- pdg %>%  
 subset(electron==T)  
  
e.bootstrap.ind <- rbind(e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind)  
  
pdg <- rbind(pdg,e.bootstrap.ind)  
pdg$index <- 1:nrow(pdg)  
  
e.bootstrap.ind <- e.bootstrap.ind$index  
  
cnn.dat <- rbind(cnn.dat,cnn.dat[e.bootstrap.ind,])  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn.rdata")  
save(pdg,file="~/Thesis data/SemiFullData/pdg.rdata")

install.packages("keras")  
  
library(keras)  
install\_keras()

load("~/Thesis data/SemiFullData/cnn.rdata")  
load("~/Thesis data/SemiFullData/pdg.rdata")  
load("~/Thesis data/SemiFullData/pdg.holdout.rdata")  
load("~/Thesis data/SemiFullData/holdout.rdata")  
require(keras)

x\_train <- array\_reshape(cnn.dat, c(nrow(cnn.dat), 264))  
x\_test <- array\_reshape(holdout.data, c(nrow(holdout.data), 264))  
  
y\_train <- as.matrix(ifelse(pdg$electron, 1,0))  
y\_test <- as.matrix(ifelse(pdg.holdout$electron, 1,0))  
  
scale.factor <- mean(mean(x\_train),mean(x\_test))  
  
x\_train <- x\_train/scale.factor  
x\_test <- x\_test/scale.factor

y\_test = to\_categorical(y\_test)  
y\_train = to\_categorical(y\_train)  
  
dim(x\_train) <- c(nrow(x\_train), ncol(x\_train))  
dim(x\_test) <- c(nrow(x\_test), ncol(x\_test))

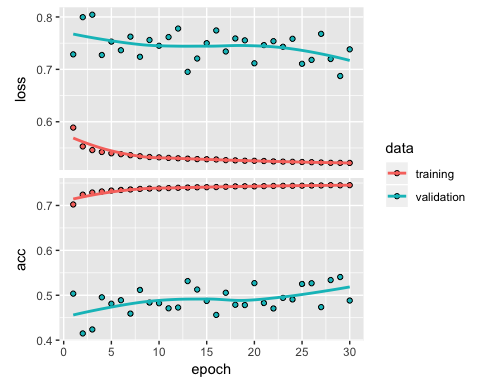
save(x\_test,file="~/Thesis data/SemiFullData/xtest.rdata")  
save(x\_train,file="~/Thesis data/SemiFullData/xtrain.rdata")  
save(y\_train,file="~/Thesis data/SemiFullData/ytrain.rdata")  
save(y\_test,file="~/Thesis data/SemiFullData/ytest.rdata")  
  
  
load("~/Thesis data/SemiFullData/ytest.rdata")  
load("~/Thesis data/SemiFullData/ytrain.rdata")

require(keras)  
rm(model)  
model <- keras\_model\_sequential()   
model %>%   
 layer\_dense(units = 128, activation = "relu", input\_shape = ncol(x\_train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 64, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 2, activation = "softmax")  
  
save(model,file="~/Thesis data/SemiFullData/model.rdata")

## <pointer: 0x0>

model %>% compile(  
 loss = "categorical\_crossentropy",  
 optimizer = optimizer\_adam(),  
 metrics = c("accuracy")  
)

history <- model %>% fit(  
 x\_train, y\_train,   
 epochs = 30, batch\_size = 128,   
 validation\_split = 0.2  
)  
  
save(history,file="~/Thesis data/SemiFullData/history.rdata")



# Truly balance classes

rm(list=ls())  
load("~/Thesis data/SemiFullData/fulljson.rdata")  
load("~/Thesis data/SemiFullData/cleanpads.rdata")  
pdg <- sapply(j,`[[`,"pdgCode")  
  
pdg <- as.numeric(pdg)  
  
pdg <- as.data.frame(pdg)  
pdg$index <- rownames(pdg)  
  
pdg$pdg <- ifelse(pdg$pdg %in% c(11,-11),T,F)  
  
names(pdg) <- c("electron","index")  
  
pdg <- pdg[,c(2,1)]  
  
cnn.dat <- data.frame(cbind(pdg,pad.df))  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn2.rdata")

load("~/Thesis data/SemiFullData/cnn2.rdata")  
  
cnn.dat <- as.matrix(cnn.dat[,-c(1,2)])  
  
zero.pads <- c()  
  
for(i in 1:nrow(cnn.dat)){  
 if(all(cnn.dat[i,]==0)){  
 zero.pads <- c(zero.pads,i)  
 }  
}  
  
cnn.dat <- cnn.dat[-zero.pads,]  
pdg <- pdg[-zero.pads,]  
  
pdg$index <- 1:length(pdg$index)  
  
save(cnn.dat,file="~/Thesis data/SemiFullData/cnn2.rdata")  
save(pdg,file="~/Thesis data/SemiFullData/pdg2.rdata")

load("~/Thesis data/SemiFullData/cnn2.rdata")  
load("~/Thesis data/SemiFullData/pdg2.rdata")  
set.seed(123)  
electron.ind <- pdg %>%  
 subset(electron==T)  
  
pion.ind <- pdg %>%  
 subset(electron!=T)  
  
electron.ind <- as.numeric(electron.ind$index)  
  
pion.ind <- as.numeric(pion.ind$index)  
  
final.test <- c(base::sample(electron.ind,100,replace = F),base::sample(pion.ind,100,replace=F))  
  
electron.ind <- electron.ind %>%  
 subset(!electron.ind %in% final.test)  
  
pion.ind <- pion.ind %>%  
 subset(!pion.ind %in% final.test)  
  
N <- length(pion.ind)  
  
electron.ind <- sample(electron.ind,N, replace = T)  
  
train.id <- c(electron.ind,pion.ind)  
train.id <- sample(train.id)  
test.id <- final.test  
  
train <- cnn.dat[train.id,]  
test <- cnn.dat[test.id,]  
  
save(train,file="~/Thesis data/SemiFullData/train.rdata")  
save(test,file="~/Thesis data/SemiFullData/test.rdata")  
  
pdg.train <- pdg[train.id,]  
pdg.test <- pdg[test.id,]  
  
save(pdg.train,file="~/Thesis data/SemiFullData/y\_train.rdata")  
save(pdg.test,file="~/Thesis data/SemiFullData/y\_test.rdata")

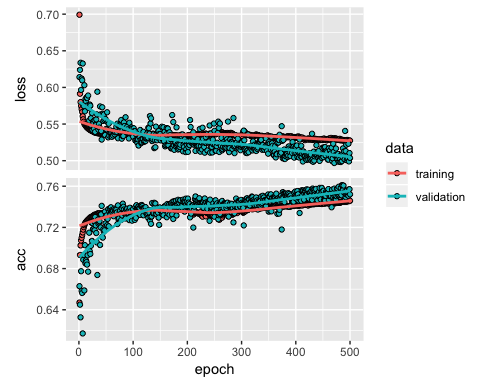
rm(list=ls())  
load("~/Thesis data/SemiFullData/train.rdata")  
load("~/Thesis data/SemiFullData/test.rdata")  
load("~/Thesis data/SemiFullData/y\_train.rdata")  
load("~/Thesis data/SemiFullData/y\_test.rdata")

require(keras)  
rm(model2)  
model2 <- keras\_model\_sequential()   
model2 %>%   
 layer\_dense(units = 256, activation = "relu", input\_shape = ncol(train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 128, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 64, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 32, activation = "relu") %>%  
 layer\_dropout(rate = 0.2) %>%  
 layer\_dense(units = 16, activation = "relu") %>%  
 layer\_dropout(rate = 0.2) %>%  
 layer\_dense(units = 2, activation = "softmax")  
  
save(model2,file="~/Thesis data/SemiFullData/model2.rdata")

model2 %>% compile(  
 loss = "categorical\_crossentropy",  
 optimizer = optimizer\_rmsprop(),  
 metrics = c("accuracy")  
)

y\_test <- as.matrix(ifelse(pdg.test$electron,1,0))  
y\_test <- to\_categorical(y\_test)  
  
y\_train <- as.matrix(ifelse(pdg.train$electron,1,0))  
y\_train <- to\_categorical(y\_train)

history2 <- model2 %>% fit(  
 train, y\_train,   
 epochs = 500, batch\_size = 1000,   
 validation\_split = 0.2  
)  
  
save(history2,file="~/Thesis data/SemiFullData/history2.rdata")  
save(model2,file="~/Thesis data/SemiFullData/model2.rdata")



load("~/Thesis data/SemiFullData/ytest.rdata")  
p <- model2 %>% predict\_proba(test)  
  
p <- cbind(as.matrix(y\_test),p)  
  
p <- p[,c(2,4)]  
  
p <-data.frame(p)  
  
names(p) <- c("actual","model2.pred")  
  
p$error <- p$actual-p$model2.pred  
  
p$squared.error <- p$error^2  
  
mean(p$squared.error)  
  
p$p0.5 <- ifelse(p$model2.pred>=0.5,1,0)  
p$p0.5 <- as.character(p$p0.5)  
p$actual <- as.character(p$actual)  
  
t1 <- table(p$actual,p$p0.5)  
  
save(t1,file="~/Thesis data/SemiFullData/t1.rdata")

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 84 | 16 |
| 1 | 38 | 62 |

p2 <- model2 %>% predict\_proba(train)  
  
p2 <- cbind(as.matrix(y\_train),p2)  
  
p2 <- p2[,c(2,4)]  
  
p2 <-data.frame(p2)  
  
names(p2) <- c("actual","model2.pred")  
  
p2$error <- p2$actual-p2$model2.pred  
  
p2$squared.error <- p2$error^2  
  
mean(p2$squared.error)  
  
p2$p0.5 <- ifelse(p2$model2.pred>=0.5,1,0)  
p2$p0.5 <- as.character(p2$p0.5)  
p2$actual <- as.character(p2$actual)  
  
t2 <- table(p2$actual,p2$p0.5)  
save(t2,file="~/Thesis data/SemiFullData/t2.rdata")

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| 0 | 514878 | 77988 |
| 1 | 205381 | 387485 |

## [1] "Our second keras deep learning model correctly detects 65.3579392307874% of electrons in our training set (bootstrapped sample size n = 1 185 732)"

## [1] "Our second keras deep learning model incorrectly classifies 13.1544058859844% of pions in our training set as electrons (bootstrapped sample size n = 1 185 732)"

## [1] "Our second keras deep learning model correctly detects 62% of electrons in our holdout set (sample size n = 100"

## [1] "Our second keras deep learning model incorrectly classifies 16% of pions in our holdout set as electrons (sample size n = 100"

//TODO:

Optimize prediction cut-off point.

# Even Deeper Learning

require(keras)  
#rm(model3)  
model3 <- keras\_model\_sequential()   
model3 %>%   
 layer\_dense(units = 256, activation = "relu", input\_shape = ncol(train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 256, activation = "relu", input\_shape = ncol(train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 256, activation = "relu", input\_shape = ncol(train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 256, activation = "relu", input\_shape = ncol(train)) %>%   
 layer\_dropout(rate = 0.4) %>%   
 layer\_dense(units = 128, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 64, activation = "relu") %>%  
 layer\_dropout(rate = 0.3) %>%  
 layer\_dense(units = 32, activation = "relu") %>%  
 layer\_dropout(rate = 0.2) %>%  
 layer\_dense(units = 16, activation = "relu") %>%  
 layer\_dropout(rate = 0.2) %>%  
 layer\_dense(units = 2, activation = "softmax")  
  
save(model3,file="~/Thesis data/SemiFullData/model3.rdata")

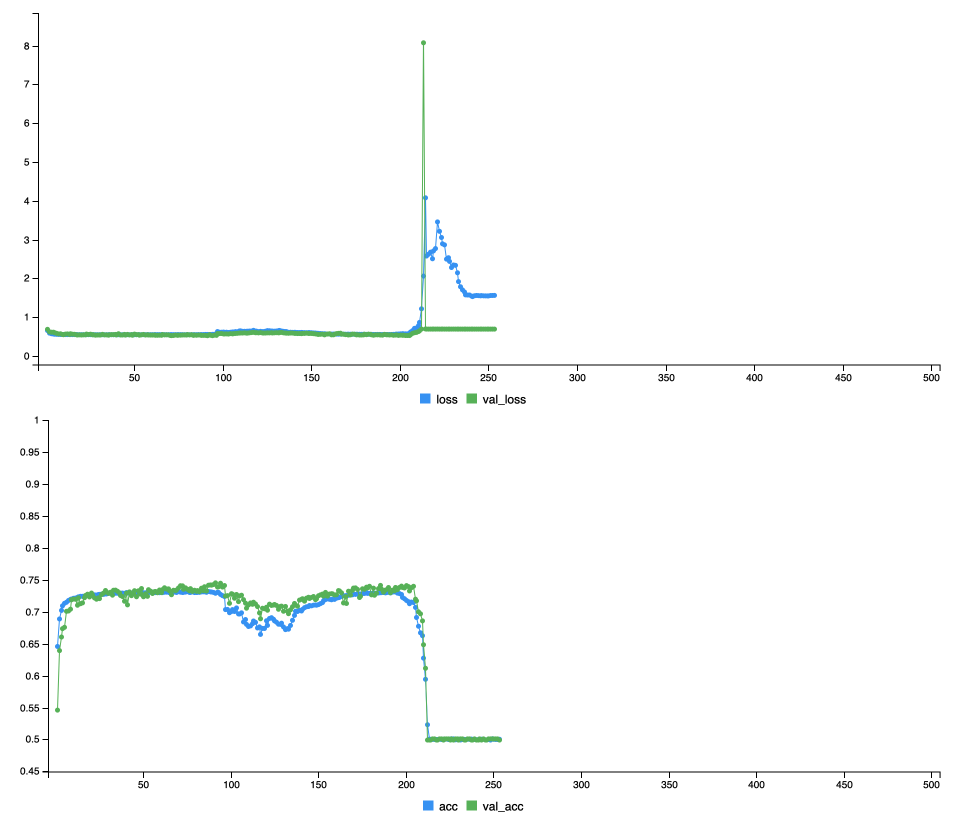
model3 %>% compile(  
 loss = "categorical\_crossentropy",  
 optimizer = optimizer\_rmsprop(),  
 metrics = c("accuracy")  
)

history3 <- model3 %>% fit(  
 train, y\_train,   
 epochs = 500, batch\_size = 1000,   
 validation\_split = 0.2  
)  
  
save(history3,file="~/Thesis data/SemiFullData/history3.rdata")  
save(model3,file="~/Thesis data/SemiFullData/model3.rdata")

## Model 3 Analysis

### Possible Causes:

* Learning rate too high
* Overfitting (too many epochs, oversampling of electrons)
* Log operation in categorical cross-entropy causing unexpected jumps in loss function, which in turn causes backpropagation to adjust weights by too much
* Highly likely that neural network started outputting a single class prediction for the last 40-50 epochs, resulting in accuracy of around 0.5 throughout these training rounds.



# Undersampling of pions, instead of oversampling of electrons

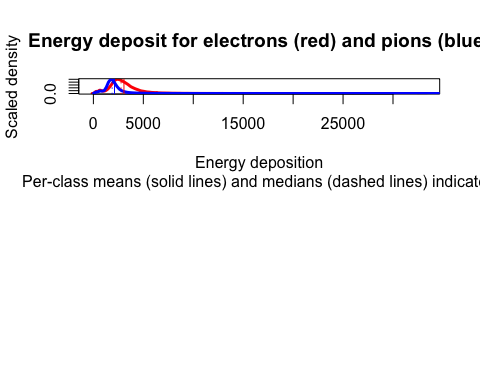
## Using the second neural network, draw a sample of pions of the same n as the sample of electrons, prefering those pions are most similar to electrons based on nusupervised learning methods

rm(list=ls())  
load("~/Thesis data/SemiFullData/fulljson.rdata")  
load("~/Thesis data/SemiFullData/cleanpads.rdata")  
pdg <- sapply(j,`[[`,"pdgCode")  
  
pdg <- as.numeric(pdg)  
  
pdg <- as.data.frame(pdg)  
pdg$index <- rownames(pdg)  
  
pdg$pdg <- ifelse(pdg$pdg %in% c(11,-11),1,0)  
  
names(pdg) <- c("electron","index")  
  
pdg <- pdg[,c(2,1)]  
  
cnn.dat <- data.frame(cbind(pdg,pad.df))

cnn.dat <- as.matrix(cnn.dat[,-c(1,2)])  
  
zero.pads <- c()  
  
for(i in 1:nrow(cnn.dat)){  
 if(all(cnn.dat[i,]==0)){  
 zero.pads <- c(zero.pads,i)  
 }  
}  
  
cnn.dat <- cnn.dat[-zero.pads,]  
pdg <- pdg[-zero.pads,]  
  
pdg$index <- 1:nrow(pdg)  
rm(zero.pads)

set.seed(1234)  
require(dplyr)  
  
elec\_ind <- pdg %>%  
 subset(electron==1) %>%  
 select(index) %>%  
 unlist() %>%  
 as.numeric()  
  
pion\_ind <- pdg %>%  
 subset(electron!=1) %>%  
 select(index) %>%  
 unlist() %>%  
 as.numeric()  
  
e.h <- sample(elec\_ind,500,replace=F)  
p.h <- sample(pion\_ind,500,replace=F)  
  
test.ind <- c(e.h,p.h)  
  
train.ind <- pdg$index[-test.ind] %>%  
 unlist() %>%  
 as.numeric()  
  
test.y <- pdg[test.ind,]  
train.y <- pdg[train.ind,]  
  
train\_electrons <- train.y %>%  
 subset(electron==1) %>%  
 select(index) %>%  
 unlist() %>%  
 as.numeric()  
  
train\_pions <- train.y %>%  
 subset(electron==0) %>%  
 select(index) %>%  
 unlist() %>%  
 as.numeric()  
  
train\_electrons <- cnn.dat[train\_electrons,]  
train\_pions <- cnn.dat[train\_pions,]  
  
save(train\_electrons,file="~/Thesis data/SemiFullData/train\_electrons.rdata")  
save(train\_pions,file="~/Thesis data/SemiFullData/train\_pions.rdata")  
save(test.y,file="~/Thesis data/SemiFullData/test.y.rdata")  
save(train.y,file="~/Thesis data/SemiFullData/train.y.rdata")

load("~/Thesis data/SemiFullData/train\_electrons.rdata")  
load("~/Thesis data/SemiFullData/train\_pions.rdata")  
e.rs <- as.numeric(rowSums(train\_electrons))  
p.rs <- as.numeric(rowSums(train\_pions))  
  
d.e.rs <- density(e.rs)  
d.p.rs <- density(p.rs)  
  
par(mfcol=c(2,1))  
  
sc.f <- length(p.rs)/length(e.rs)  
  
plot(x=d.e.rs$x,y=d.e.rs$y/max(d.e.rs$y),col="red",type="l",lwd=3,main="Energy deposit for electrons (red) and pions (blue)",xlab="Energy deposition",ylab="Scaled density",sub="Per-class means (solid lines) and medians (dashed lines) indicated")  
lines(x=d.p.rs$x,y=d.p.rs$y/max(d.p.rs$y),col="blue",lwd=3)  
abline(v=median(e.rs),col="red",lty="dotted")  
abline(v=median(p.rs),col="blue",lty="dotted")  
abline(v=mean(e.rs),col="red")  
abline(v=mean(p.rs),col="blue")



elec.feat <- data.frame(e.rs)  
pion.feat <- data.frame(p.rs)  
  
rm(e.rs)  
rm(p.rs)  
  
names(elec.feat) <- "energy\_deposit"  
names(pion.feat) <- "energy\_deposit"  
  
range.lo <- 0  
range.hi <- max(range(train\_electrons)[2],range(train\_pions)[2])+100  
  
breaks <- seq(range.lo,range.hi,50)  
  
e.bins <- apply(train\_electrons,1,function(x) hist(x,breaks = breaks, plot=F)$count)  
  
p.bins <- apply(train\_pions,1,function(x) hist(x,breaks = breaks, plot=F)$count)  
  
rm(train\_electrons)  
rm(train\_pions)  
  
e.bins <- t(e.bins)  
p.bins <- t(p.bins)  
  
elec.feat <- data.frame(cbind(elec.feat,e.bins))  
pion.feat <- data.frame(cbind(pion.feat,p.bins))  
  
save(elec.feat,file="~/Thesis data/SemiFullData/elec.feat.rdata")  
save(pion.feat,file="~/Thesis data/SemiFullData/pion.feat.rdata")

# Fuzzy Clustering

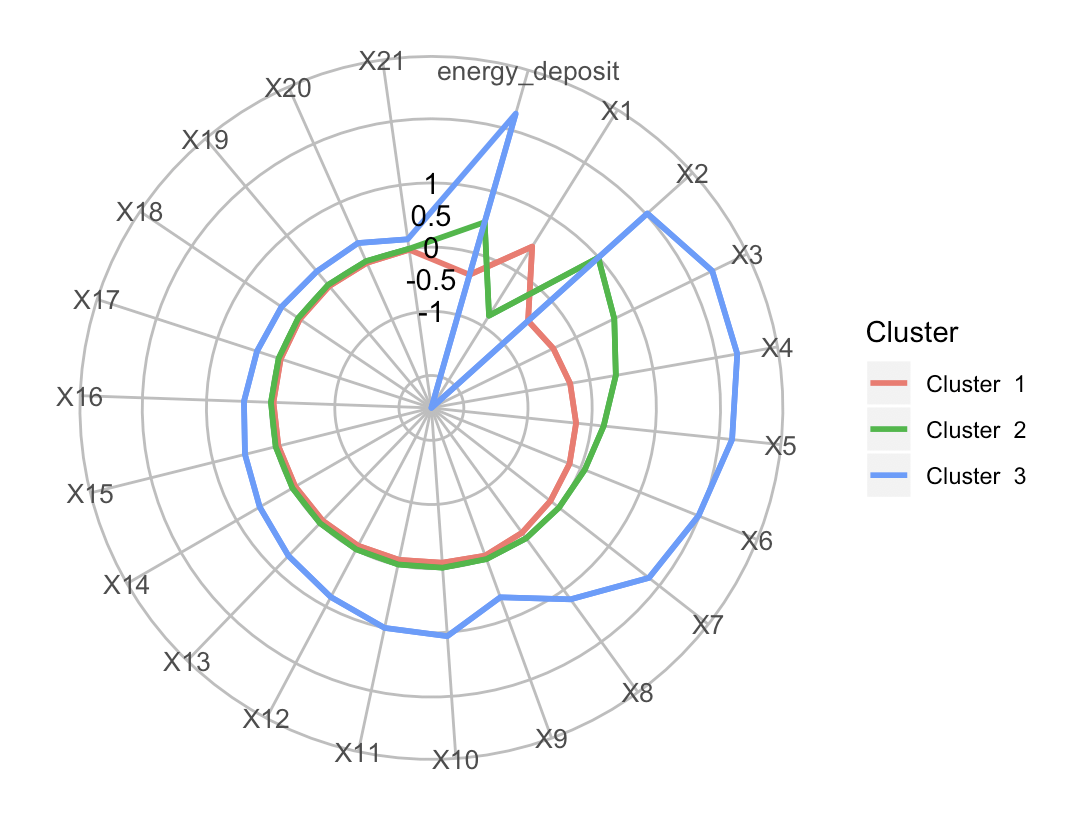
## Fuzzy C-means

Mathematical distance computed with euclidian norm:

load("~/Thesis data/SemiFullData/elec.feat.rdata")  
load("~/Thesis data/SemiFullData/pion.feat.rdata")  
cl.d <- rbind(elec.feat,pion.feat)  
  
all(cl.d[,23]==0)  
  
cl.d <- cl.d[,-23]  
  
cl.d <- scale(cl.d)  
  
rm(elec.feat)  
rm(pion.feat)

require(advclust)  
fuzclust <- fuzzy.CM(X=cl.d,K = 3,m = 2,RandomNumber = 1234)  
  
save(fuzclust,file="~/Thesis data/SemiFullData/fuzclust.rdata")  
  
load("~/Thesis data/SemiFullData/train.y.rdata")  
  
unsupervised.features <- data.frame(cbind(train.y,fuzclust@member))  
  
save(unsupervised.features,file="~/Thesis data/SemiFullData/unsup\_feat1.rdata")

radar.plotting(fuzclust, cl.d)

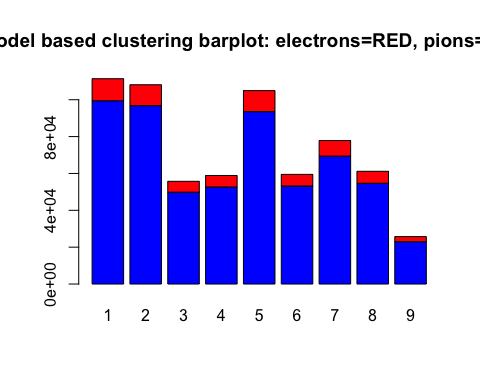


Radarplot for c means clustering

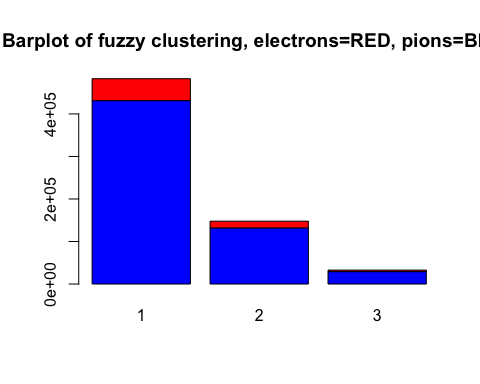
## Model based clustering

require(mclust)  
  
clustered.dat <- Mclust(cl.d)  
  
save(clustered.dat,file="~/Thesis data/SemiFullData/mclust.rdata")  
  
unsupervised.features <- data.frame(cbind(unsupervised.features,clustered.dat$z))  
  
save(unsupervised.features,file="~/Thesis data/SemiFullData/unsup\_feat1.rdata")

load("~/Thesis data/SemiFullData/fuzclust.rdata")  
load("~/Thesis data/SemiFullData/mclust.rdata")  
load("~/Thesis data/SemiFullData/unsup\_feat1.rdata")  
barplot(table(as.character(unsupervised.features$electron),as.character(clustered.dat$classification)),col=c("blue","red"),  
 main="Model based clustering barplot: electrons=RED, pions=BLUE")



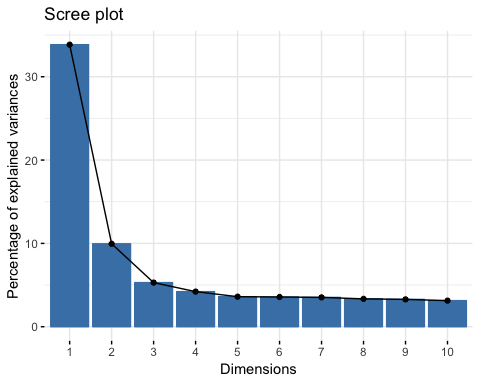
barplot(table(as.character(unsupervised.features$electron), as.character(fuzclust@hard.label)),main="Barplot of fuzzy clustering, electrons=RED, pions=BLUE",col=c("blue","red"))



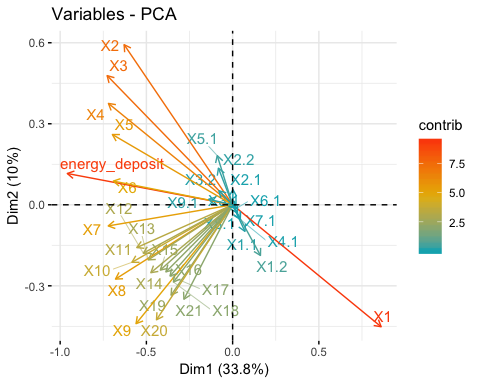
cl.d.2 <- data.frame(cbind(cl.d,unsupervised.features))  
  
  
  
save(cl.d.2,file="~/Thesis data/SemiFullData/cld2.rdata")

load("~/Thesis data/SemiFullData/cld2.rdata")  
  
cl.d.2 <- cl.d.2[,-c(23:24)]  
  
pc <- prcomp(cl.d.2)

require(factoextra)  
fviz\_eig(pc)

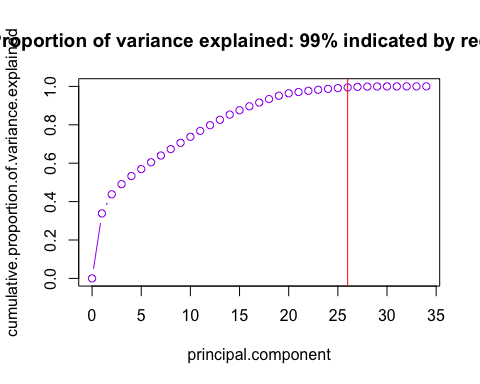


fviz\_pca\_var(pc,  
 col.var = "contrib", # Color by contributions to the PC  
 gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  
 repel = TRUE # Avoid text overlapping  
 )



find.uncert.pions.x <- pc$x

principal.component <- 0:34  
cumulative.proportion.of.variance.explained <- c(0,cumsum(pc$sdev^2)/sum(pc$sdev^2))  
  
plot(principal.component,cumulative.proportion.of.variance.explained,type="b",col="purple",main="Proportion of variance explained: 99% indicated by red line")  
abline(v=min(which(cumulative.proportion.of.variance.explained>=.99)),col="red")

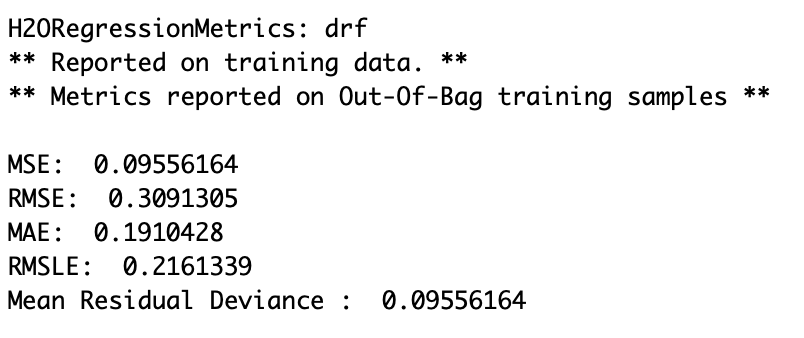


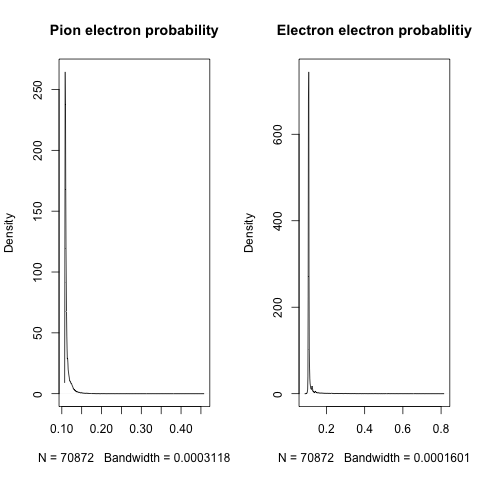
# H2O

require(h2o)  
h2o.init()

load("~/Thesis data/SemiFullData/train.y.rdata")  
  
dat <- data.frame(cbind(find.uncert.pions.x,train.y$electron))  
names(dat)[35] <- "electron"  
  
dat.hex <- as.h2o(dat)  
  
find.uncert.pion <- h2o.randomForest(x=1:34,y=35,dat.hex,nfolds=10)

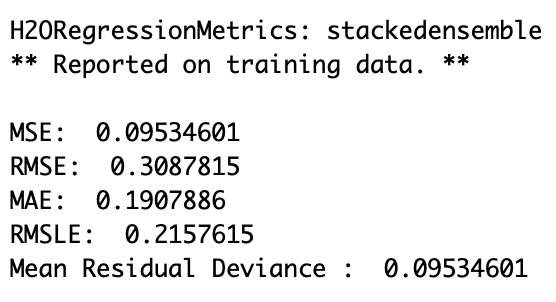
h2o.performance(find.uncert.pion)  
  
p.rf <- h2o.predict(find.uncert.pion,dat.hex)  
  
p.rf <- as.data.frame(p.rf)  
  
p.rf <- data.frame(cbind(train.y,p.rf))  
  
row.names(p.rf) <- NULL  
  
p.rf$index <- 1:nrow(p.rf)  
  
electron.sample.ind <- p.rf$index[train.y$electron==1]  
  
pion.sample.ind <- p.rf[train.y$electron==0,]  
  
pion.sample.ind <- pion.sample.ind[order(pion.sample.ind$predict,decreasing = T),]  
  
N <- length(electron.sample.ind)  
  
pion.sample.ind <- pion.sample.ind$index[1:N]  
  
any(is.na(pion.sample.ind))  
  
  
  
png(filename="~/Thesis data/SemiFullData/probability.dens.png")  
par(mfrow=c(1,2))  
  
d <- p.rf[pion.sample.ind,]  
  
d <- as.numeric(d$predict)  
  
max(d)  
plot(density(d),"Pion electron probability")  
  
d <- p.rf[electron.sample.ind,]  
  
d <- as.numeric(d$predict)  
  
max(d)  
plot(density(d),"Electron electron probablitiy")  
  
dev.off()  
  
save(pion.sample.ind,file="~/Thesis data/SemiFullData/pion.sample.ind.rdata")  
save(electron.sample.ind,file="~/Thesis data/SemiFullData/electron.sample.ind.rdata")

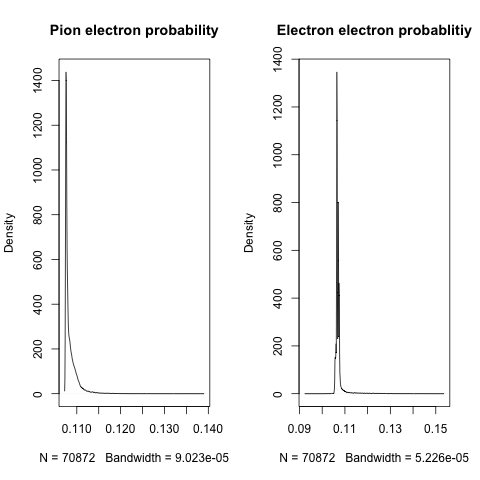




find.uncert.pion.2 <- h2o.automl(x=1:34,y=35,dat.hex,nfolds=10)

h2o.performance(find.uncert.pion.2@leader)  
  
p.rf <- h2o.predict(find.uncert.pion.2@leader,dat.hex)  
  
p.rf <- as.data.frame(p.rf)  
  
p.rf <- data.frame(cbind(train.y,p.rf))  
  
row.names(p.rf) <- NULL  
  
p.rf$index <- 1:nrow(p.rf)  
  
electron.sample.ind <- p.rf$index[train.y$electron==1]  
  
pion.sample.ind <- p.rf[train.y$electron==0,]  
  
pion.sample.ind <- pion.sample.ind[order(pion.sample.ind$predict,decreasing = T),]  
  
N <- length(electron.sample.ind)  
  
pion.sample.ind <- pion.sample.ind$index[1:N]  
  
any(is.na(pion.sample.ind))  
  
  
  
png(filename="~/Thesis data/SemiFullData/probability.dens2.png")  
par(mfrow=c(1,2))  
  
d <- p.rf[pion.sample.ind,]  
  
d <- as.numeric(d$predict)  
  
max(d)  
plot(density(d),"Pion electron probability")  
  
d <- p.rf[electron.sample.ind,]  
  
d <- as.numeric(d$predict)  
  
max(d)  
plot(density(d),"Electron electron probablitiy")  
  
dev.off()  
  
save(pion.sample.ind,file="~/Thesis data/SemiFullData/pion.sample.ind.rdata")  
save(electron.sample.ind,file="~/Thesis data/SemiFullData/electron.sample.ind.rdata")





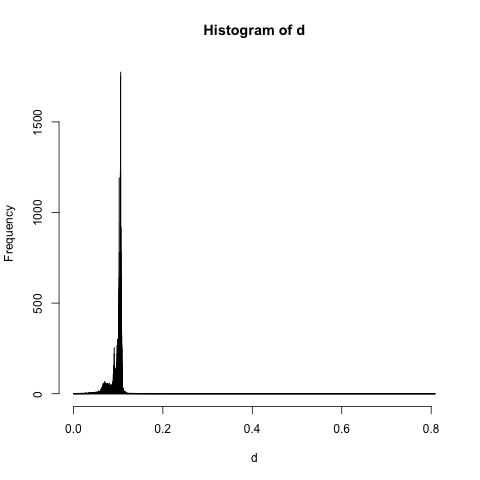
dat$electron <- as.factor(dat$electron)  
  
dat2.hex <- as.h2o(dat)  
  
find.uncert.pion.classification <- h2o.automl(x=1:34,y=35,dat2.hex,nfolds=10)

h2o.performance(find.uncert.pion.classification@leader)  
  
h2o.confusionMatrix(find.uncert.pion.classification@leader)  
  
p.rf <- h2o.predict(find.uncert.pion.classification,dat.hex)  
  
p.rf <- as.data.frame(p.rf$p1)  
  
p.rf <- data.frame(cbind(train.y,p.rf))  
  
row.names(p.rf) <- NULL  
  
p.rf$index <- 1:nrow(p.rf)  
  
electron.sample.ind <- p.rf$index[train.y$electron==1]  
  
pion.sample.ind <- p.rf[train.y$electron==0,]  
  
pion.sample.ind <- pion.sample.ind[order(pion.sample.ind$p1,decreasing = T),]  
  
N <- length(electron.sample.ind)  
  
pion.sample.ind <- pion.sample.ind$index[1:N]  
  
any(is.na(pion.sample.ind))  
  
  
  
png(filename="~/Thesis data/SemiFullData/probability.dens3.png")  
par(mfrow=c(1,2))  
  
d <- p.rf[pion.sample.ind,]  
  
d <- as.numeric(d$p1)  
  
max(d)  
plot(density(d),"Pion electron probability")  
  
d <- p.rf[electron.sample.ind,]  
  
d <- as.numeric(d$p1)  
  
max(d)  
plot(density(d),"Electron electron probablitiy")  
  
dev.off()  
  
save(pion.sample.ind,file="~/Thesis data/SemiFullData/pion.sample.ind.rdata")  
save(electron.sample.ind,file="~/Thesis data/SemiFullData/electron.sample.ind.rdata")

Training on principal components results in very similar predictions for all observations, rather train on features created for clustering algorithms above.

rm(list=ls())  
load("~/Thesis data/SemiFullData/cld2.rdata")  
load("~/Thesis data/SemiFullData/train.y.rdata")  
  
cl.d.2 <- cl.d.2[,-c(23,24)]  
  
dat <- data.frame(cbind(cl.d.2,train.y$electron))  
  
dat$train.y.electron <- ifelse(dat$train.y.electron==1,"electron","pion")  
  
dat$train.y.electron <- as.factor(dat$train.y.electron)  
  
h2o.shutdown(prompt = F)  
  
h2o.init()  
  
h2o.removeAll()  
  
dat.hex <- as.h2o(dat,"dat.hex")  
  
rf.m <- h2o.randomForest(x=1:34,y=35,training\_frame="dat.hex",nfolds=10,  
 ntrees = 100,balance\_classes = T,  
 stopping\_metric = "AUC",stopping\_tolerance = 0.001)  
  
p <- h2o.predict(rf.m,dat.hex)  
  
p <- as.data.frame(p)  
  
a <- train.y  
  
p <- data.frame(cbind(a,p))  
  
h2o.performance(rf.m)  
  
save(p,file="~/Thesis data/SemiFullData/p.rdata")  
  
p$predict <- ifelse(as.character(p$predict)=="electron",1,0)  
  
save(p,file="~/Thesis data/SemiFullData/p.rdata")

load("~/Thesis data/SemiFullData/electron.sample.ind.rdata")  
  
N <- length(electron.sample.ind)  
  
pion.sample.ind <- p[p$electron==0,]  
  
pion.sample.ind <- pion.sample.ind[order(pion.sample.ind$electron.1,decreasing = T),]  
  
pion.sample.ind <- pion.sample.ind$index[1:N]  
  
png(filename="~/Thesis data/SemiFullData/probability.dens4.png")  
par(mfrow=c(1,1))  
  
d <- p[pion.sample.ind,]  
  
d <- as.numeric(d$electron.1)  
  
hist(d,breaks=10000)  
  
dev.off()  
  
save(pion.sample.ind,file="~/Thesis data/SemiFullData/pion.sample.ind.rdata")



# Convolutional Neural Networks

rm(list=ls())  
load("~/Thesis data/SemiFullData/train\_electrons.rdata")  
load("~/Thesis data/SemiFullData/train\_pions.rdata")  
load("~/Thesis data/SemiFullData/train.y.rdata")  
  
x\_train <- rbind(train\_electrons,train\_pions)  
  
load("~/Thesis data/SemiFullData/pion.sample.ind.rdata")  
load("~/Thesis data/SemiFullData/electron.sample.ind.rdata")  
  
final.ind <- c(electron.sample.ind,pion.sample.ind)

x\_train <- cnn.dat[final.ind,]  
y\_train <- as.matrix(pdg$electron[final.ind])  
y\_train <- to\_categorical(y\_train)

require(keras)  
  
batch\_size <- 1000  
num\_classes <- 2  
epochs <- 100  
  
img\_rows <- 11  
img\_cols <- 24  
  
x\_train <- scale(x\_train)  
  
x\_train <- as.array(x\_train)  
  
x\_train <- array\_reshape(x\_train, c(nrow(y\_train), img\_rows, img\_cols,1))  
  
input\_shape <- c(img\_rows, img\_cols, 1)  
  
  
  
model <- keras\_model\_sequential() %>%  
 layer\_conv\_2d(filters = 32, kernel\_size = c(3,3), activation = 'relu',  
 input\_shape = input\_shape) %>%   
 layer\_conv\_2d(filters = 64, kernel\_size = c(3,3), activation = 'relu') %>%   
 layer\_max\_pooling\_2d(pool\_size = c(2, 2)) %>%   
 layer\_dropout(rate = 0.25) %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 128, activation = 'relu') %>%   
 layer\_dropout(rate = 0.5) %>%   
 layer\_dense(units = num\_classes, activation = 'softmax')  
  
summary(model)  
  
# Compile model  
model %>% compile(  
 loss = loss\_categorical\_crossentropy,  
 optimizer = optimizer\_adadelta(),  
 metrics = c('accuracy')  
)  
  
# Train model  
model %>% fit(  
 x\_train, y\_train,  
 batch\_size = batch\_size,  
 epochs = epochs,  
 validation\_split = 0.2  
)  
  
  
  
  
scores <- model %>% evaluate(  
 x\_test, y\_test, verbose = 0  
)  
  
# Output metrics  
cat('Test loss:', scores[[1]], '\n')  
cat('Test accuracy:', scores[[2]], '\n')