GERHARD VILJOEN

PARTICLE IDENTIFICATION ML

```
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:

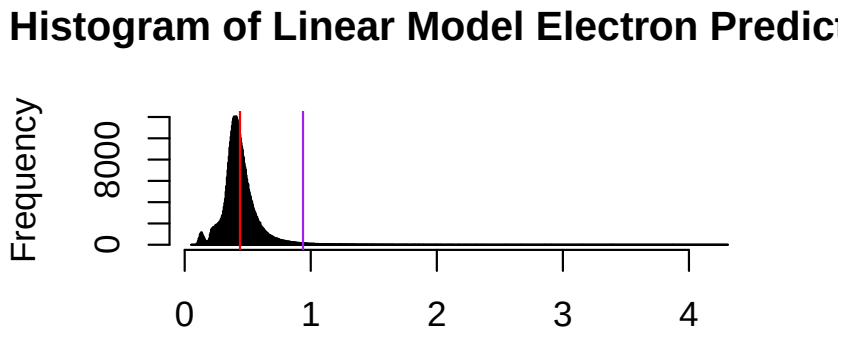
```
linmod1 <- lm(electron~.,data=train)

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$`Linear Model Prediction`,breaks=1000,main="Histogram of Linear Model Electron Prediction",s
abline(v=mean(accuracy$`Linear Model Prediction`),col="red")
abline(v=q,col="purple")
```



Mean prediction (red), and 99th quantile (purple) indic

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
```

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
```

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=-

q <- optim.q$par
```

```
## [1] "After optimizing the electron prediction cut-off, by maximizing the area under the curve of the re
```

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
```

```
## [1] "Based on predictions from our cut-off optimized basic linear model, we incorrectly classify 24.972
```

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
```

```
## [1] "Based on predictions from our cut-off optimized basic linear model, we correctly accept 66.85513%
```

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")
```

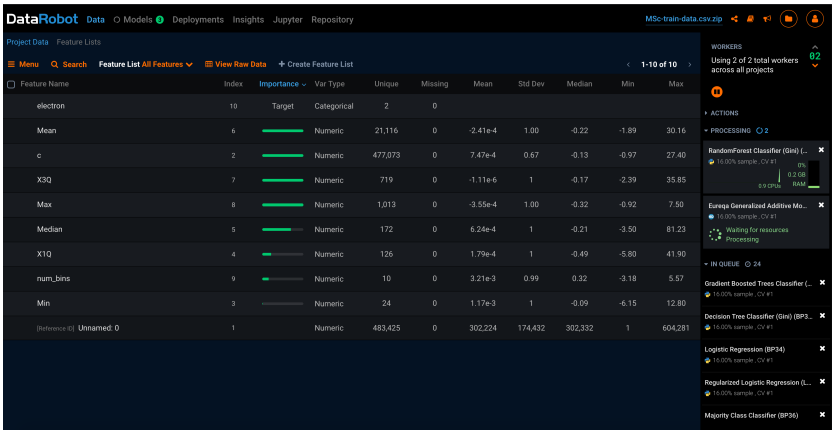


Figure 1: Figure1: Feature Importance

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

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

The ROC curve and Prediction Distribtution plots are as follows:

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

```
## [1] "DataRobot detects 16.9139694010917% of electrons in the holdout set it created from the FULL datas
```

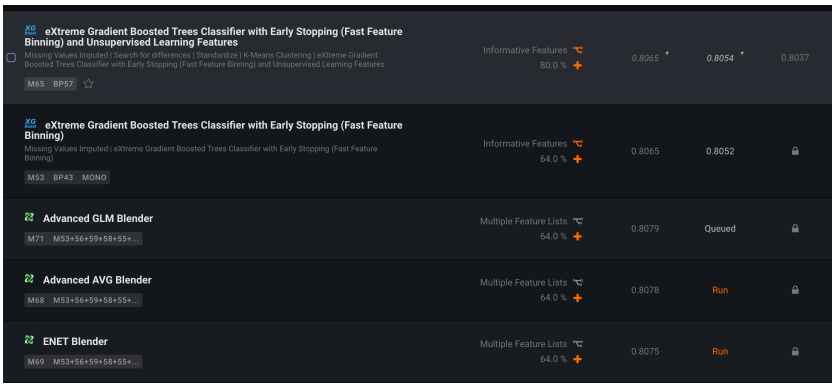


Figure 2: Figure 2: Top 5 DataRobot models

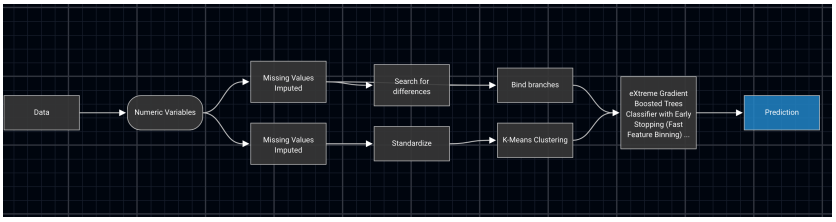


Figure 3: Figure 3: Extreme Gradient Boosted Tree Classifier with Early Stopping (Fast Feature Binning) Bueprint

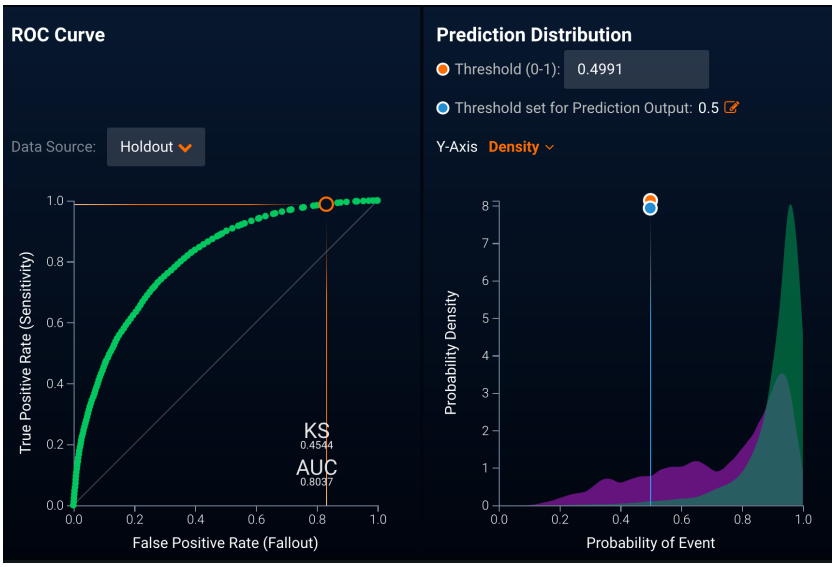


Figure 4: Figure 4: Top Model Evaluation

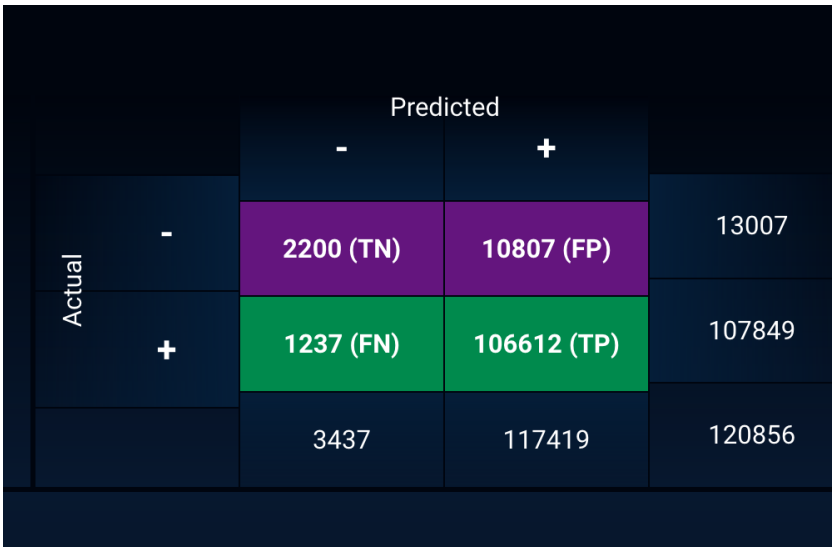


Figure 5: Figure 5: Top Model Confusion Matrix

```
## [1] "DataRobot incorrectly classifies 1.14697400995837% of pions as electrons in the holdout set it cre
```

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:

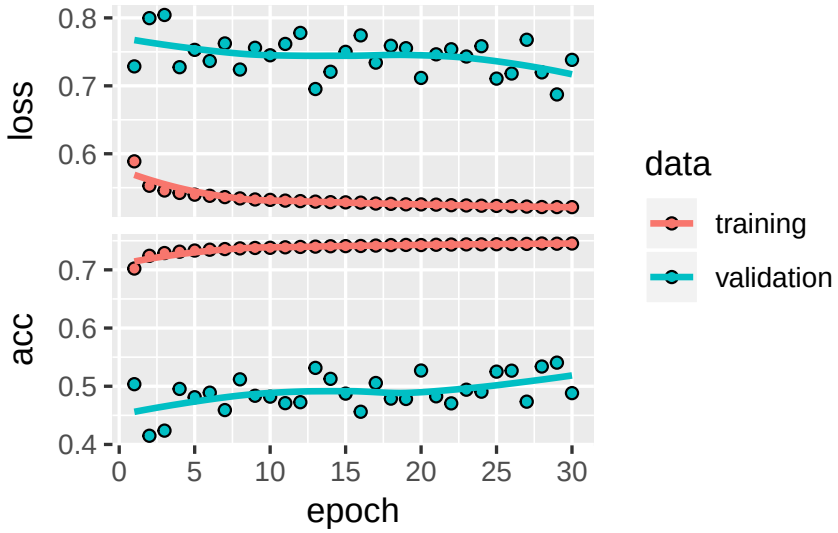
```
e.bootstrap.ind <- pdg %>%
  subset(electron==T)

e.bootstrap.ind <- rbind(e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e.bootstrap.ind,e

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")
```

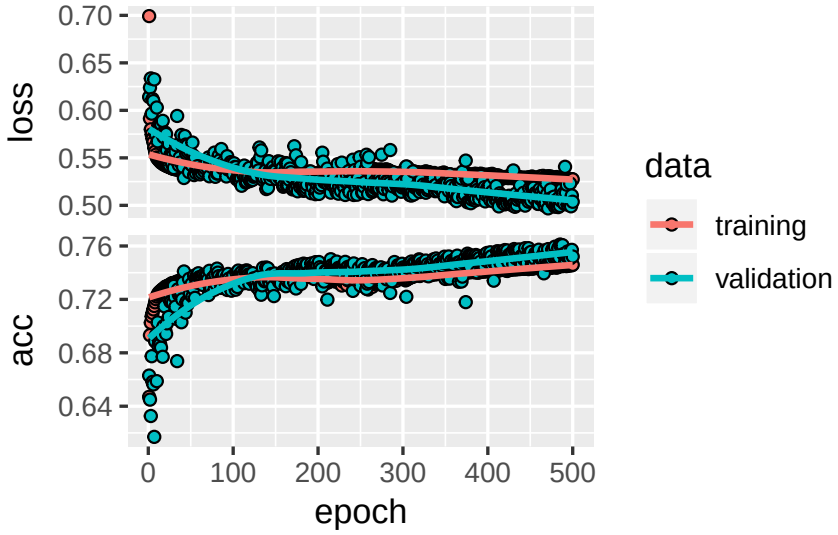


Truly balance classes

```
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 train

## [1] "Our second keras deep learning model incorrectly classifies 13.1544058859844% of pions in our trai

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

## [1] "Our second keras deep learning model incorrectly classifies 16% of pions in our holdout set as ele
```

//TODO:
Optimize prediction cut-off point.

Even Deeper Learning

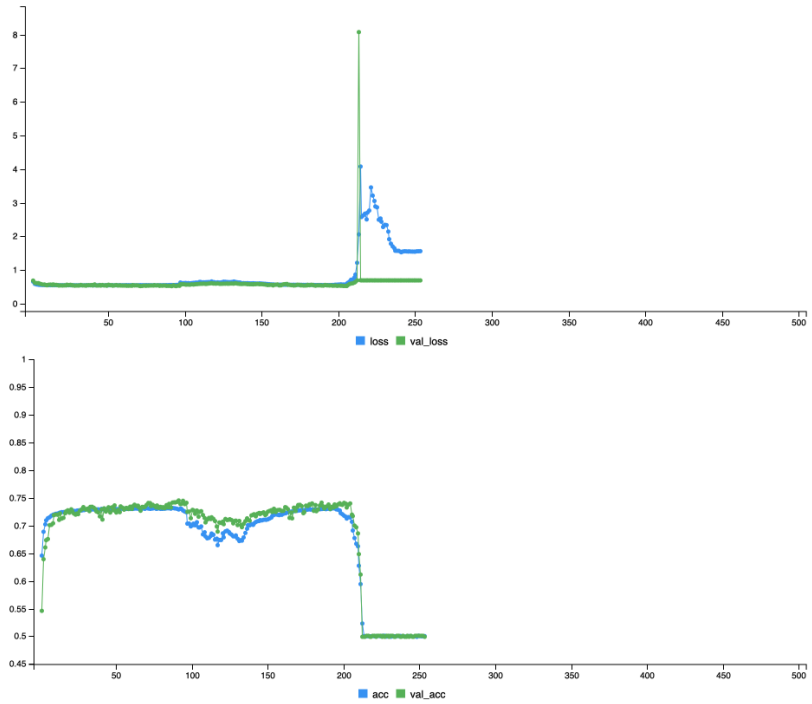
```
require(keras)
#rm(model3)
model3 <- keras_model_sequential()
model3 %>%
  layer_dense(units = 256, activation = "relu", input_shape = ncol(train)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 256, activation = "relu", input_shape = ncol(train)) %>%
  layer_dropout(rate = 0.4) %>%
  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.



Convolutional Neural Networks