1. Data Exploration

Introduction

In 2014, CERN provided a simulated dataset from their ATLAS experiment for use in a programming competition on Kaggle. After the Kaggle Challenge closed, the full dataset was made available here, with the accompanying documentation here. The goal of the challenge is to develop a binary classification model to distinguish signal and background events.

The dataset contains 818,238 samples with 30 features, a class label, and 4 columns of additional information such as sample weight and event ID. Of this, 250,000 samples were provided as training data for the Kaggle Challenge (set "t"), 100,000 for the public leaderboard (set "b"), 450,000 for the private leaderboard (set "v"), and the remaining 18,238 were unused (set "u").

The samples are simulated events from the Large Hadron Collider (LHC). In an event, bunches of protons are accelerated around the LHC in opposite directions and collide. The collision produces hundreds of particles, most of which are unstable and decay into lighter particles such as electrons or photons. Sensors in the LHC measure properties of the surviving particles, and from this the properties of the parent particles can be inferred. Signal events are defined as events where a Higgs boson decays into two tau particles.

In the following markdown notebooks we use methods from SM1 and SC1 to attempt the Kaggle Challenge in R. We chose to follow a similar structure, using the Kaggle training set to train our model, and the private leaderboard set as our hold-out validation set. The public leaderboard and unused datasets are excluded for simplicity.

The notebooks are structured as follows:

1. Data Exploration - an introduction to the dataset, its structure, and our R package `lhc`.
2. SVM - we attempt to use our implementation of SVM algorithms to train a classifier, though we are restricted to train on a small fraction of the data.
3. Logistic Regression - we use our implementation of logistic regression within an OOP framework to explore models with varying parameters and feature engineering.
4. Results - we compare our models and use the best performing model on our validation set.

The package we have developed for assessment is called `lhc` and is available on github here. The functions are divided thematically into the following files:

- utility_funcs.r
- objects.r
- plot_funcs.r
- lr_funcs.r
- kernel_funcs.r
- svm_funcs.r
- project_funcs.r

Data Exploration

Load packages

```
devtools::install_github("ant-stephenson/lhc")
```

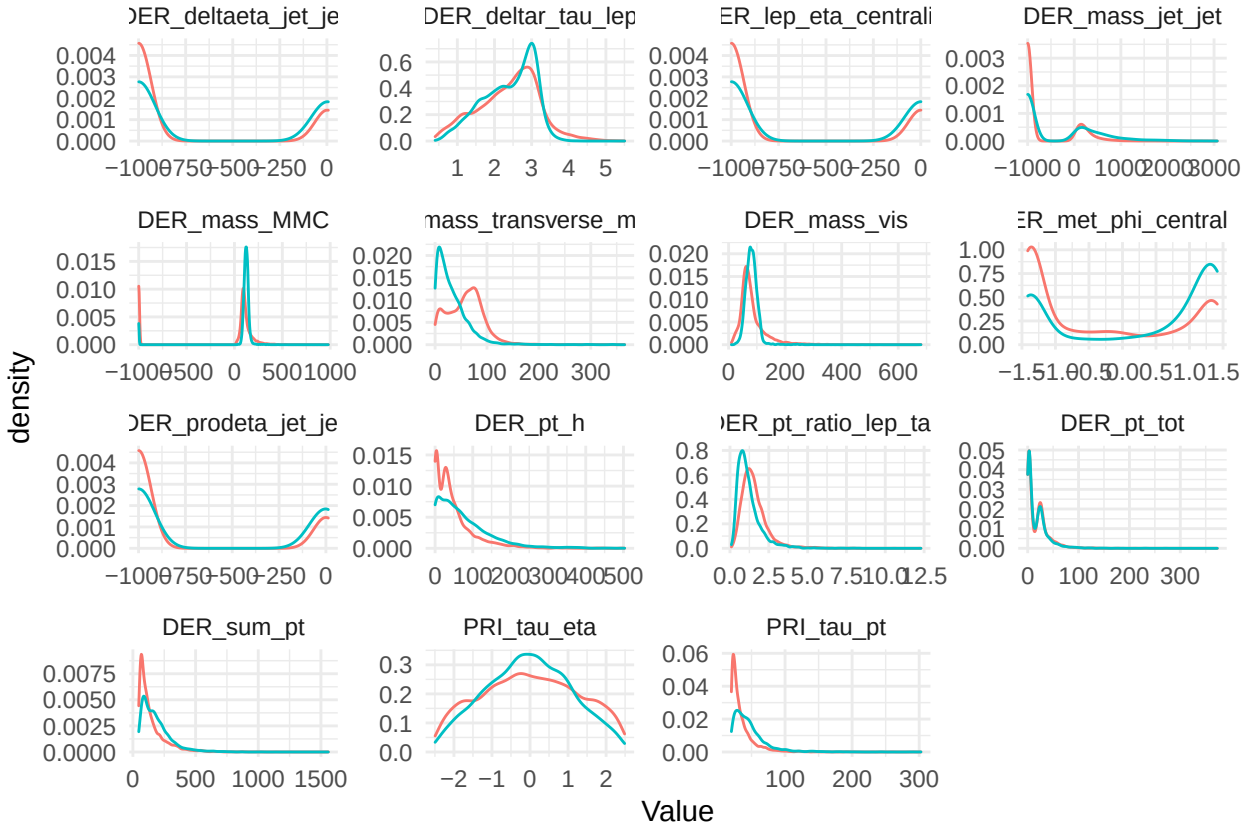
```
##      checking for file '/tmp/RtmpGqrUZY/remotes3e976564245b/ant-stephenson-lhc-ba72fc9/DESCRIPTION' ...
##   -  preparing 'lhc':
##    checking DESCRIPTION meta-information ...  v  checking DESCRIPTION meta-information
```

```
##   -  checking for LF line-endings in source and make files and shell scripts
##   -  checking for empty or unneeded directories
## -  building 'lhc_0.1.0.tar.gz'
##
##
```

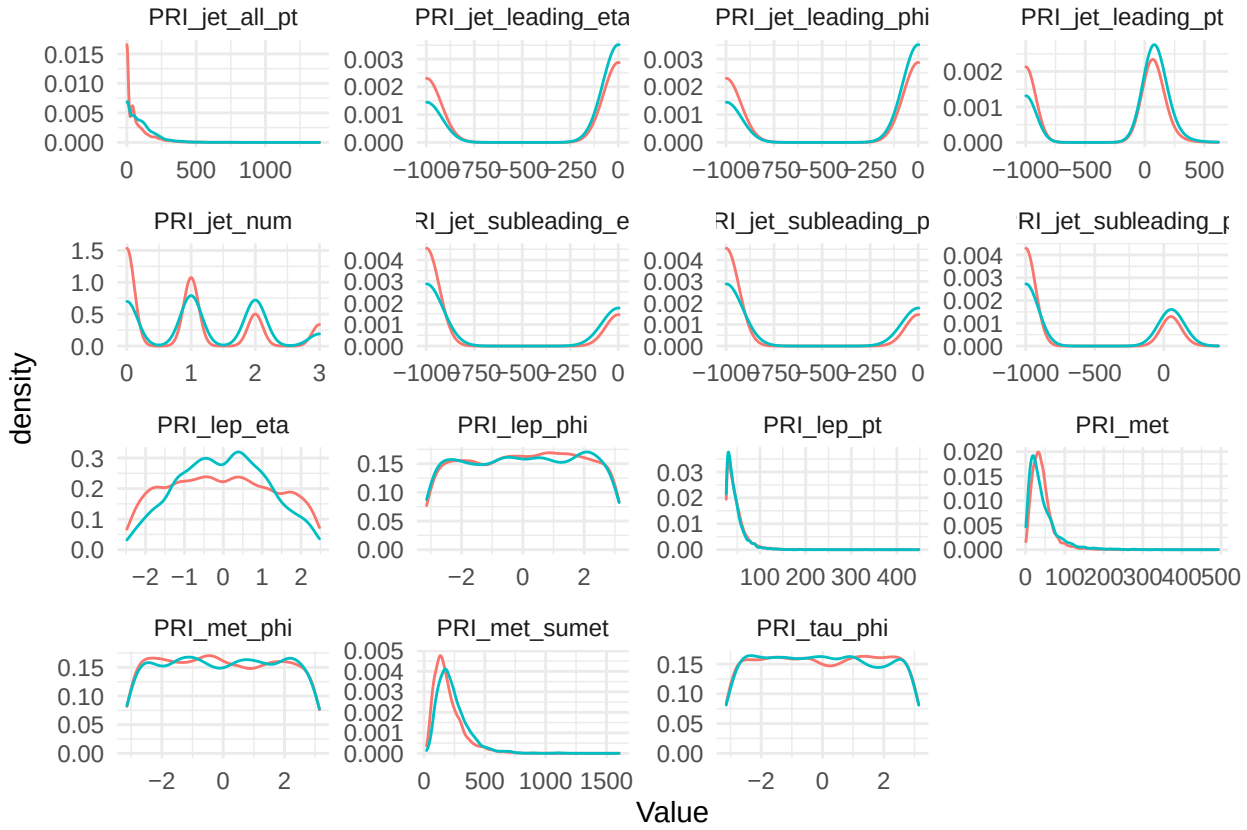
```r
library(lhc)
library(dplyr)
library(ggplot2)
library(tidyr)
library(kableExtra)
```

Load dataset

```r
#get the filepath of the dataset
filepath <- list.files(path="~", pattern="atlas-higgs-challenge-2014-v2.csv",
                       full.names=T, recursive=T)

#fast load the raw data
raw_data <- data.table::fread(filepath)

#Split data into X (just variables) and additional info
all_info <- as.data.frame(raw_data[, c("KaggleSet", "KaggleWeight", "Weight", "Label")])
all_X <- as.matrix(raw_data[, -c("EventId", "KaggleSet", "KaggleWeight", "Weight", "Label")])

#assign event id as row names so we can check the two stay matched
rownames(all_info) <- rownames(all_X) <- raw_data$EventId

#add a column with numerical coding of class label (0,1)
all_info$Y <- as.numeric(all_info$Label == "s")

#Select the training set
X <- all_X[raw_data$KaggleSet=="t",]
info <- all_info[raw_data$KaggleSet=="t",]

#View the variables
plot_distributions(X, variables = colnames(X)[1:15], labels = info$Label)
```




```
plot_distributions(X, variables = colnames(X)[16:30], labels = info$Label)
```



By looking at the distributions of the data we can see that the two classes appear fairly similar and so we might expect a reasonably complex model will be needed to separate them. All of the variables are continuous except for

PRI_jet_num which is discrete (0, 1, 2, 3). For some of the variables there are peaks at -999. This is because -999 has been used to indicate undefined values which cannot be computed for a physical reason.

Performance Metrics

The simulated dataset has been generated so that the number of signal and background events are roughly equal. However in reality, there are far more background events than signal, and so the samples have also been given importance weightings. The weightings within each class sum to the actual expected number of signal and background events, $\sum_{i;y_i=s} w_i = N_s$, and $\sum_{i;y_i=b} w_i = N_b$. Whenever we take a training and test set from our total training data, we need to ensure the weightings are rescaled.

To assess the accuracy of a binary classification model, a standard approach is to plot a ROC (Receiver Operating Characteristic) curve and calculate its AUC (Area Under the Curve). To create the plot, you take the output of a probabilistic model and plot the true positive rate against the false positive rate as the decision threshold is varied. Using this approach here will tell us about the model's accuracy in terms of number of samples in the dataset, but it will not take into account the sample weights. Note that this approach cannot be used for non-probabilistic models such as SVM, as there is not a continuous output that different thresholds can be applied to.

The objective of the Kaggle Challenge is instead to maximise the AMS (Approximate Median discovery Significance) metric, which is defined as:

$$\text{AMS} = \sqrt{2\left(\left(s+b+b_{reg}\right)\ln\left(1+\frac{s}{b+b_{reg}}\right)-s\right)}$$

where s is the sum of the sample weights of true positives $s = \sum_{i;y_i=s,\hat{y}_i=s} w_i$, and b is the sum of sample weights of false positives $b = \sum_{i;y_i=b,\hat{y}_i=s} w_i$, and $b_{reg} = 10$. For probabilistic models we can look at how the AMS changes as different decision thresholds are applied, and we can use it to select the most appropriate threshold.

In our package lhc, we have defined two reference class objects to store data related to the ROC and AMS performance measures ROC_curve and AMS_data. We have also defined plotting functions to plot multiple ROC curves or AMS metrics on the same axes in order to visualise the variance of models during cross validation.

```
#when we take a subset of samples, weights need to be renormalised so the sum of signal weights = Ns and s
Ns <- sum(info$KaggleWeight[info$Label=="s"])
  #or sum(all_info$Weight[all_info$Label=="s"]) as the two are equal
Nb <- sum(info$KaggleWeight[info$Label=="b"])
  #or sum(all_info$Weight[all_info$Label=="b"])

#these values are set as the default scaling factors in our function ams_metric
#which is called within the AMS_data class
print(c(Ns, Nb))
```

```
## [1]    691.9886 410999.8473
```

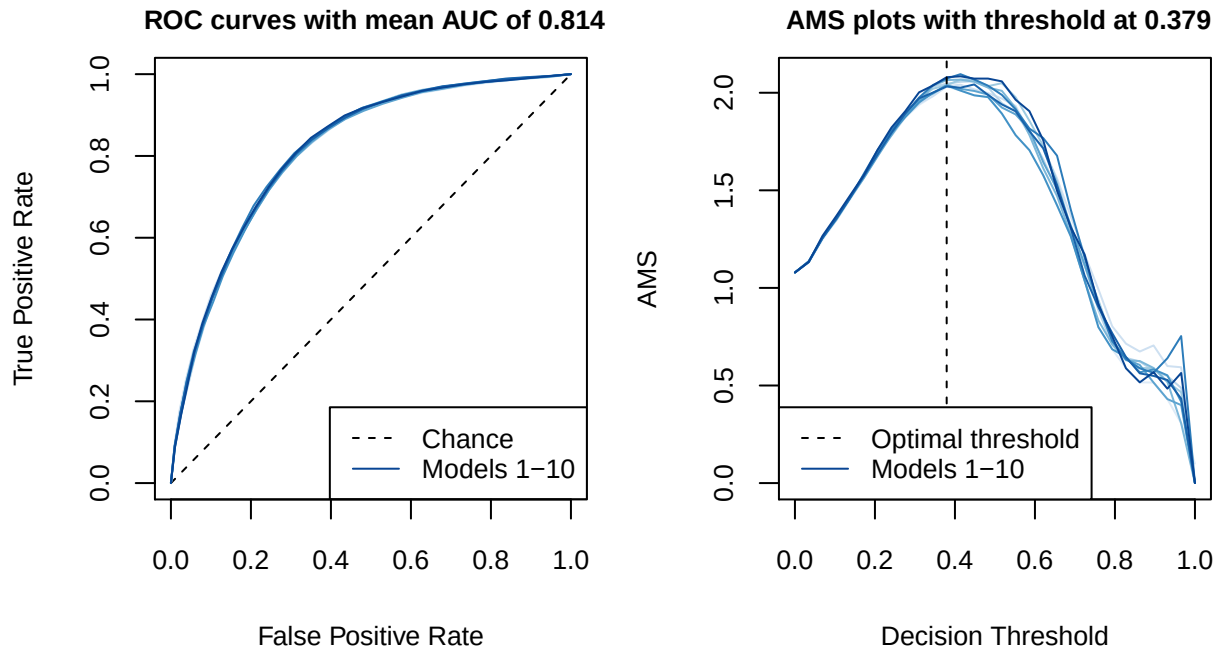
Baseline Model

In our package, we have implemented logistic regression with iteratively weighted least squares (IWLS) in the function logistic_reg, and we have created an associated object class logistic_model.

As a first pass, we perform a simple logistic regression on the training set with k-fold cross validation, and view the ROC curves and AMS data for each fold.

```
# get an index for CV groups
k <- 10
kI <- partition_data(n=nrow(X), k=k, random=T)

#create lists to hold the k models, roc and ams data
models <- vector("list", k)
rocs <- vector("list", k)
amss <- vector("list", k)
```

```
#for each fold, subset the training and test data
for(i in 1:k){
  X_train <- X[kI != i,]
  y_train <- info[kI != i, "Y"]

  X_test <- X[kI == i,]
  y_test <- info[kI == i, "Y"]
  w_test <- info[kI == i, "Weight"]

  #fit a logistic regression model to the CV training data
  models[[i]] <- logistic_model$new(X=X_train, y=y_train)

  #use it to predict the classifications of the test data
  prob <- models[[i]]$predict(X_test)

  #store roc and ams data
  rocs[[i]] <- ROC_curve$new(y_test, prob)
  amss[[i]] <- AMS_data$new(y_test, prob, w_test)
}

par(mfrow=c(1,2))
par(mar=c(4,4,2,1))
plot_rocs(rocs, cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
plot_amss(amss, cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
```



We can see the logistic regression model has low variance between folds and a reasonable average AUC.

On the left of the AMS graph (t=0) all samples are classified at signal events, and so the sum of true positive weights, s, is at a maximum and the sum of false positive weights, b, is at a minimum. On the right of the graph (t=1) all samples are classified as background events, and so `s=b=0` and AMS = 0. The best decision threshold for this initial model is around t=0.4.

Missing data

Currently the undefined values are still coded as -999. Since the missing values have a physical meaning, we may expect there to be some structure to the missing values.

```r
#creating a copy of X just coding if the value is missing or non missing
missing <- matrix(as.numeric(X == -999), ncol=ncol(X))
colnames(missing) <- colnames(X)

#considering just the columns that contain missing data
missing <- missing[,colSums(missing) != 0]

#using dplyr to group the different types of row
missing_pattern <- as_tibble(missing) %>%
  group_by_all() %>%
  count() %>%
  ungroup()

#display the table over two rows with kable
display_table <- function(data){
  n <- ncol(data)/6
  for(i in 1:n){
    print(kable(data[(6*i-5):(6*i)], booktabs=T)  %>%
      kable_styling(latex_options="scale_down"))
  }
}
display_table(missing_pattern)
```

DER_mass_MMC	DER_deltaeta_jet_jet	DER_mass_jet_jet	DER_prodeta_jet_jet	DER_lep_eta_centrality	PRI_jet_leading_pt
0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	1	1	0
1	1	1	1	1	1

PRI_jet_leading_eta	PRI_jet_leading_phi	PRI_jet_subleading_pt	PRI_jet_subleading_eta	PRI_jet_subleading_phi	n
0	0	0	0	0	68114
0	0	1	1	1	69982
1	1	1	1	1	73790
0	0	0	0	0	4429
0	0	1	1	1	7562
1	1	1	1	1	26123

By looking for patterns of missing data, we identify 6 different types of sample. The first types and the last 3 types are identical other than the variable `DER_mass_MMC` being missing or not missing. By looking at the variables names, the main structure appears to be related to jets (a narrow cone of hadrons).

```r
#looking at how the missing patterns relate to the categorical variable
missing <- cbind(X["PRI_jet_num"], missing)
colnames(missing)[1] <- "PRI_jet_num"

#ignoring DER_mass_MMC and grouping again
missing_pattern <- as_tibble(missing) %>%
  select(-DER_mass_MMC) %>%
  group_by_all() %>%
  count() %>%
  ungroup()

display_table(missing_pattern)
```

PRI_jet_num	DER_deltaeta_jet_jet	DER_mass_jet_jet	DER_prodeta_jet_jet	DER_lep_eta_centrality	PRI_jet_leading_pt
0	1	1	1	1	1
1	1	1	1	1	0
2	0	0	0	0	0
3	0	0	0	0	0

PRI_jet_leading_eta	PRI_jet_leading_phi	PRI_jet_subleading_pt	PRI_jet_subleading_eta	PRI_jet_subleading_phi	n
1	1	1	1	1	99913
0	0	1	1	1	77544
0	0	0	0	0	50379
0	0	0	0	0	22164

So we have found the following pattern:

- when jet_num = 0 the following variables are undefined

```
missing_pattern[,1] <- NULL
colnames(missing_pattern)[missing_pattern[1,] == 1]
```

```
##  [1] "DER_deltaeta_jet_jet"   "DER_mass_jet_jet"       "DER_prodeta_jet_jet"
##  [4] "DER_lep_eta_centrality" "PRI_jet_leading_pt"     "PRI_jet_leading_eta"
##  [7] "PRI_jet_leading_phi"    "PRI_jet_subleading_pt"  "PRI_jet_subleading_eta"
## [10] "PRI_jet_subleading_phi"
```

- when jet_num = 1 the following variables are undefined

```
colnames(missing_pattern)[missing_pattern[2,] == 1]
```

```
## [1] "DER_deltaeta_jet_jet"   "DER_mass_jet_jet"       "DER_prodeta_jet_jet"
## [4] "DER_lep_eta_centrality" "PRI_jet_subleading_pt"  "PRI_jet_subleading_eta"
## [7] "PRI_jet_subleading_phi"
```

- and when jet_num = 2 or 3, there are no undefined variables.

This pattern makes physical sense, because if there are 0 jets, all the jet variables are missing, and if there is 1 jet all the leading jet variables are there and the subleading jet variables are missing.

Groups

It appears that splitting out the data based on patterns of missing data may be appropriate. Each of the subsets will contain similar types of event and we can remove the missing variables within each group. If we split the data into the 6 different missing data patterns, some of the groups where `DER_mass_MMC` is missing are very small (only 4,000 out of 250,000 samples). Therefore to keep our training groups sufficiently large, we will split the data into 3 groups based on `PRI_jet_num`, and aim to select 3 different models.

```
#adding a new column in info which indicates the groupings
info$Group <- factor(X["PRI_jet_num"],
                     levels=c(0, 1, 2, 3),
                     labels=c("j=0", "j=1", "j=2+", "j=2+"))

G <- nlevels(info$Group)
groups <- levels(info$Group)

#define which columns we can remove from each subset (now constants, sd=0)
features_to_rm <- vector("list", 3)
for(g in 1:G){
  features_to_rm[[g]] <- colnames(X)[apply(X[info$Group==groups[g],], 2, sd) == 0]
}
```

```
#check how the number of signal/background events are distributed in each group
n_stats <- as.data.frame(unclass(table(info$Group, info$Label)))
n_ratio <- n_stats/rowSums(n_stats)

n_stats <- cbind(n_stats, rowSums(n_stats))
n_stats <- rbind(n_stats, colSums(n_stats))
colnames(n_stats)[3] <- rownames(n_stats)[4] <- "total"

kable(n_ratio, booktabs=T)
```

	b	s
j=0	0.7448580	0.2551420
j=1	0.6426545	0.3573455
j=2+	0.5524723	0.4475277

```
kable(n_stats, booktabs=T)
```

	b	s	total
j=0	74421	25492	99913
j=1	49834	27710	77544
j=2+	40078	32465	72543
total	164333	85667	250000

```
#check how the weights are distributed
w_stats <- info %>%
  group_by(Group) %>%
  summarise(b = sum(Weight[Label=="b"]),
            s = sum(Weight[Label=="s"]),
            .groups ="drop") %>%
  as.data.frame()

rownames(w_stats) <- w_stats[,"Group"]
w_stats <- w_stats[,2:3]
w_ratio <- w_stats / rowSums(w_stats)

w_stats <- cbind(w_stats, rowSums(w_stats))
w_stats <- rbind(w_stats, colSums(w_stats))
colnames(w_stats)[3] <- rownames(w_stats)[4] <- "total"

kable(w_ratio, booktabs=T)
```

	b	s
j=0	0.9986757	0.0013243
j=1	0.9978769	0.0021231
j=2+	0.9965697	0.0034303

```
kable(w_stats, booktabs=T)
```

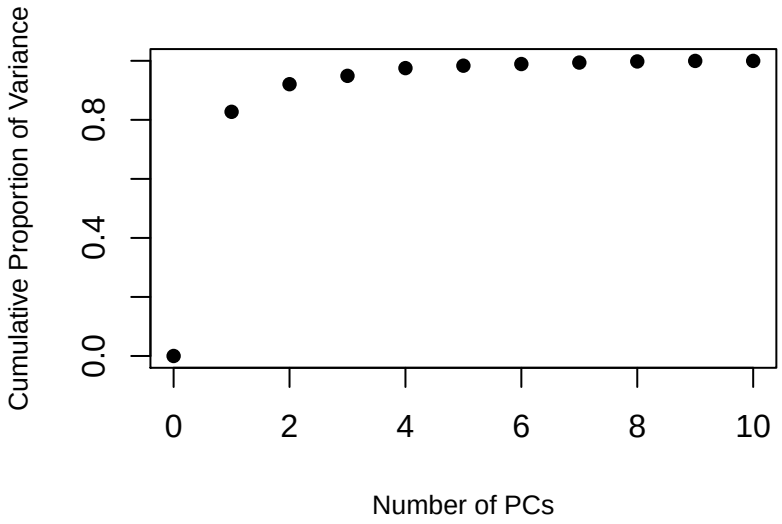
	b	s	total
j=0	85253.41	113.04725	85366.45
j=1	29315.73	62.37262	29378.10
j=2+	10748.76	36.99816	10785.76
total	125317.90	212.41803	125530.32

The three groups have varying ratios of signal and background events (approx 1:3, 1:2, and 1:1), though they have similar ratio of sample weights (approx 1:500) and have similar number of samples (70,000 to 100,000).

PCA

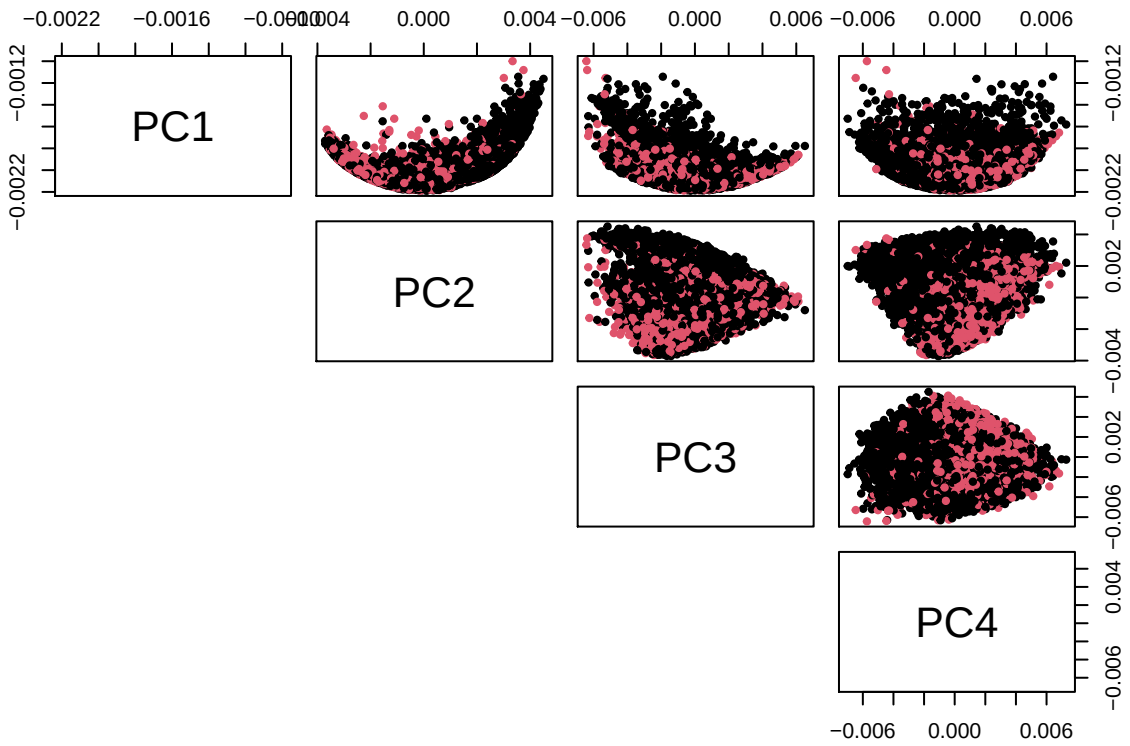
A standard approach for visualising a high dimensional dataset is to perform Principle Component Analysis (PCA) and to plot the first few principle components. We can select the number of PCs to visualise by looking at the proportion of variance that they explain.

```
#perform pca excluding -999
X_temp <- X
X_temp[X==-999] <- NA
pca <- prcomp(na.omit(t(X_temp)), scale.=T, center=T)

#plot variance explained over n pcs
var_explained <- c(0, summary(pca)$importance[3,])
plot(0:10, var_explained[1:11], xlab="Number of PCs",
     ylab="Cumulative Proportion of Variance", pch=16, cex.lab=0.8)
```



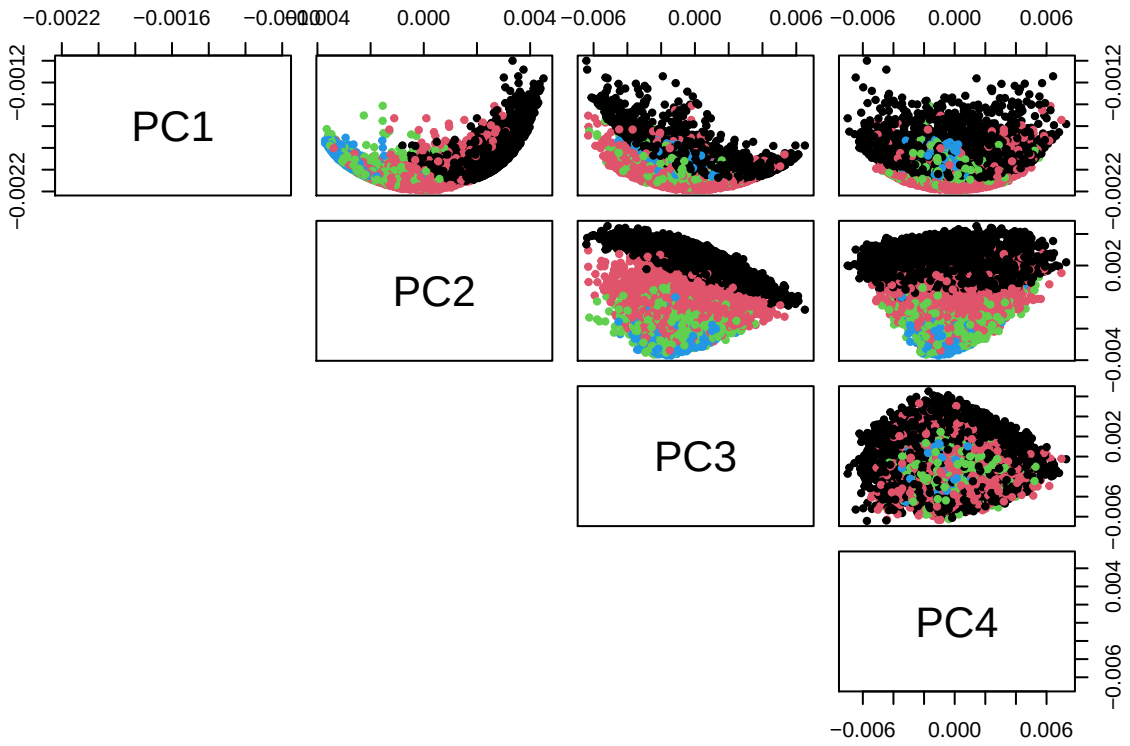
From the plot above we can see that the first 4 principle components explain nearly all the variance of the data. We can plot the transformed data coloured by class label to see if the PCA has separated the classes.

```
#the first 5 pcs explain nearly all the variance of the data
X_transformed <- pca$rotation

#plot the pcs of a random subset of samples
idx <- sample(1:nrow(X), 10000)
pairs(X_transformed[idx, 1:4], lower.panel = NULL, pch=20, col=info[idx, "Y"]+1)
```

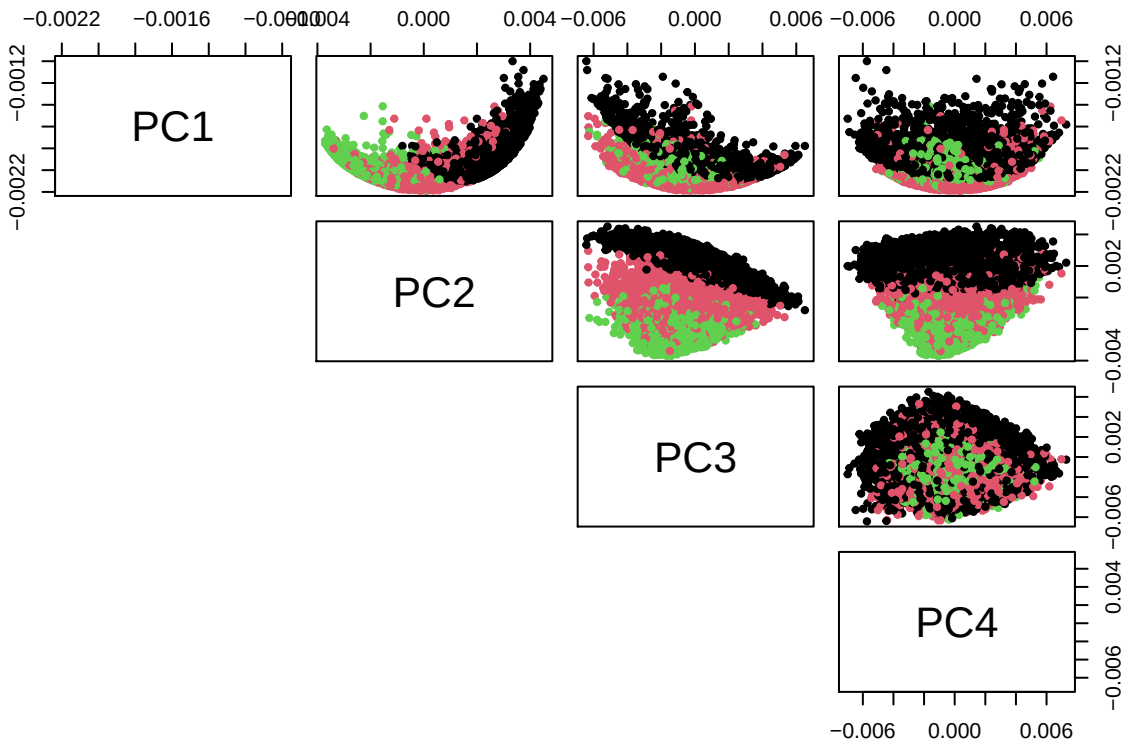


It looks like there may be some separation of classes in PC4. Let's instead colour the points by our new jet number groups.

```
#colour by the jet num
pairs(X_transformed[idx,1:4], lower.panel = NULL, pch=20, col=X[idx, "PRI_jet_num"]+1)
```




```
#colour by our groups
pairs(X_transformed[idx,1:4], lower.panel = NULL, pch=20, col=as.numeric(info[idx, "Group"]))
```

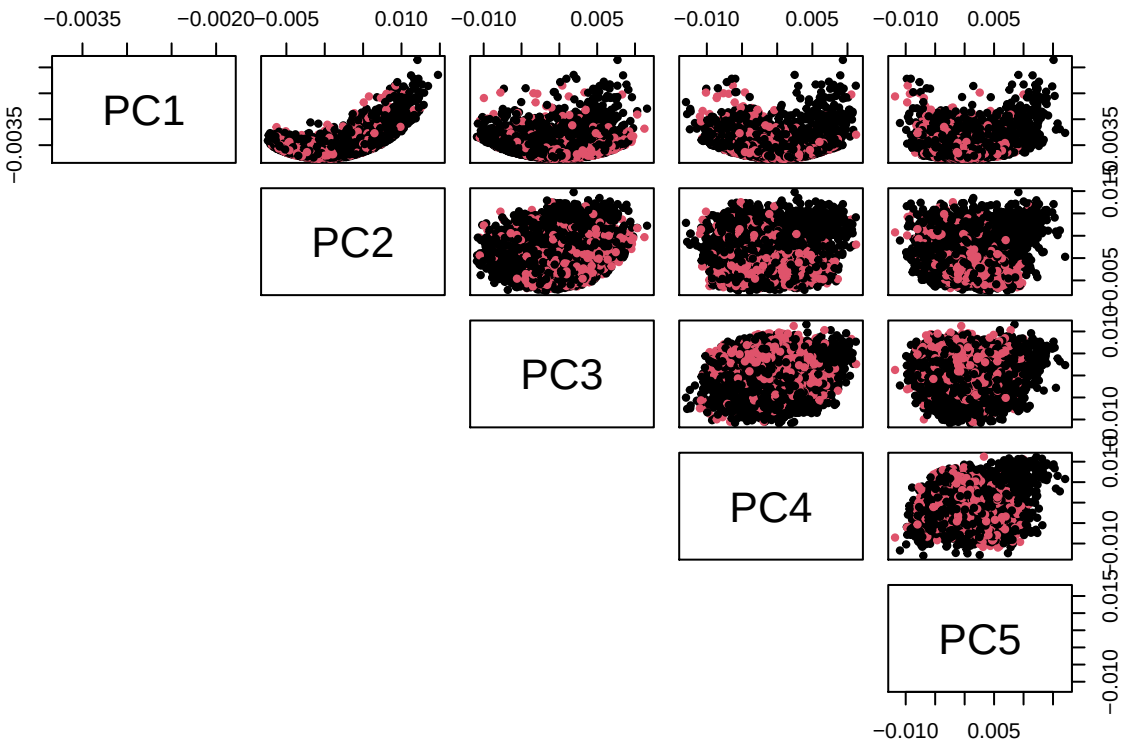



```
#its clear that the missing data is the main thing influencing the pca
#and jet num = 2 or 3 are not similar
```

PC2 appears to separate the samples based on their missing data pattern. Events with 2 or 3 jets look very similar and so having these grouped together as jet=2+ seems appropriate.

Now lets look at a PCA on just one of the groups, which should now be much less effected by missing data. Unfortunately, the samples are still not easily separable.

```
X_temp <- X_temp[info$Group=="j=1", !colnames(X) %in% features_to_rm[[1]]]
info_temp <- info[info$Group=="j=1",]

pca <- prcomp(na.omit(t(X_temp)), scale.=T, center=T)
X_transformed <- pca$rotation

idx <- sample(1:nrow(X_temp), 10000)
pairs(X_transformed[idx,1:5], lower.panel = NULL, pch=20, col=info_temp[idx, "Y"]+1)
```

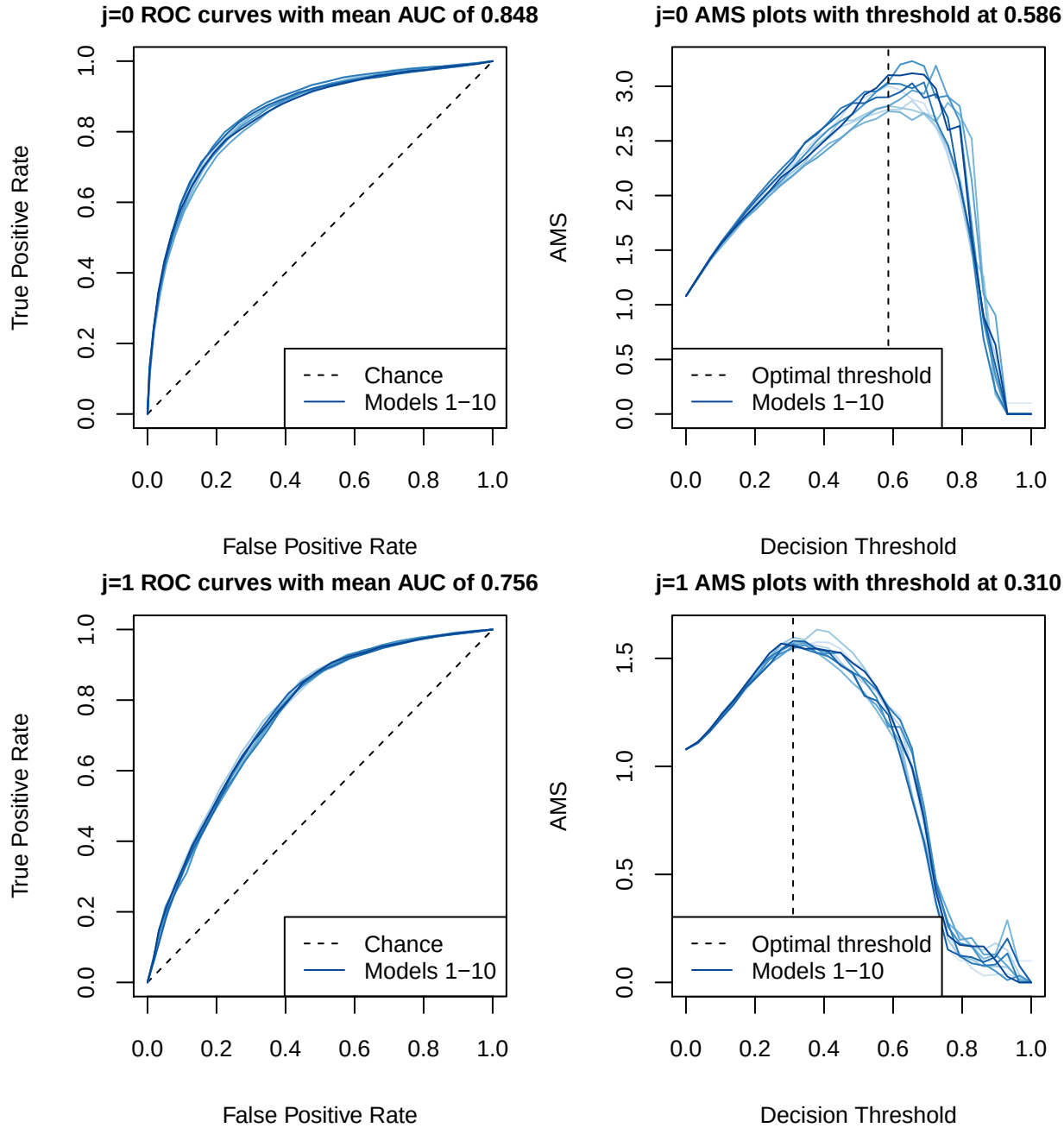


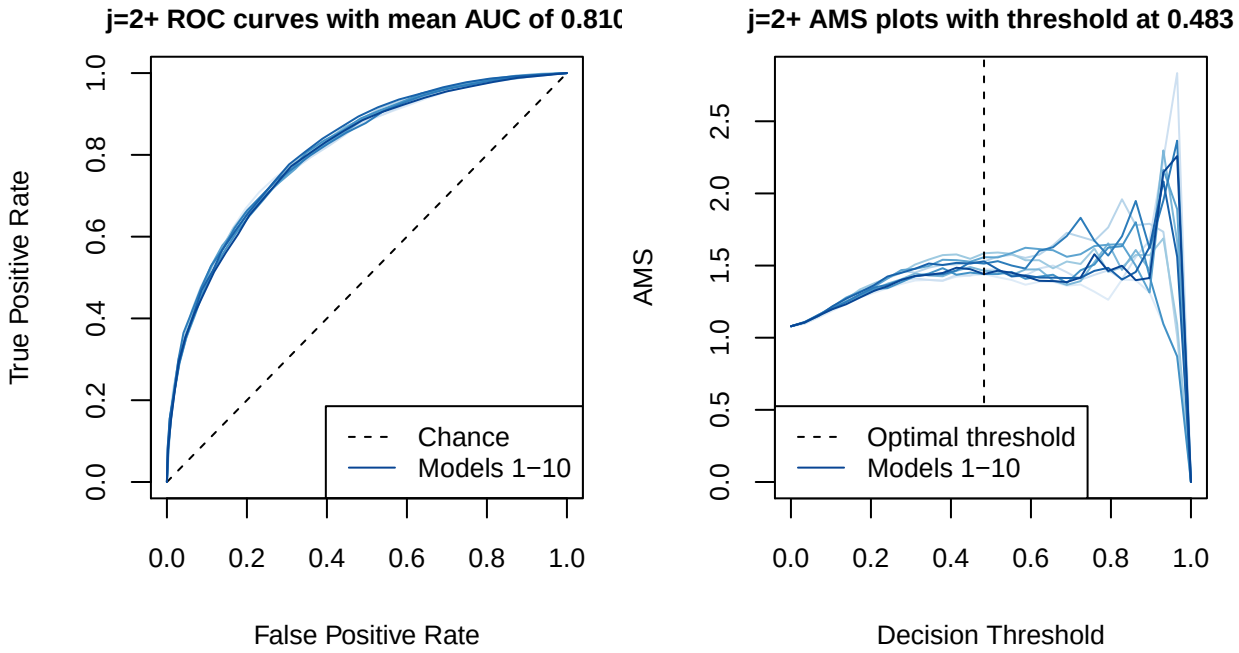
Baseline model per group

Now we have split the data into 3 groups, we'll train a logistic regression model for each group. Again we will use k-fold CV and our reference class objects.

```
par(mfrow=c(1,2))
par(mar=c(4,4,2,1))

for(g in 1:G){
  X_group <- X[info$Group==groups[g], !colnames(X) %in% features_to_rm[[g]]]
  info_group <- info[info$Group==groups[g],]

  # get an index for CV groups
  k <- 10
  kI <- partition_data(n=nrow(X_group), k=k, random=T)

  #create lists to hold the k models and k roc curves
  models <- vector("list", k)
  rocs <- vector("list", k)
  amss <- vector("list", k)

  for(i in 1:k){
    X_train <- X_group[kI != i,]
    y_train <- info_group[kI != i, "Y"]

    X_test <- X_group[kI == i,]
    y_test <- info_group[kI == i, "Y"]
    w_test <- info_group[kI == i, "Weight"]

    #fit a logistic regression model to the CV training data
    models[[i]] <- logistic_model$new(X=X_train, y=y_train)

    #use it to predict the classifications of the test data
```

```
    prob <- models[[i]]$predict(X_test)

    #store roc and ams data
    rocs[[i]] <- ROC_curve$new(y_test, prob)
    amss[[i]] <- AMS_data$new(y_test, prob, w_test)
  }

  plot_rocs(rocs, info=groups[g], cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
  plot_amss(amss, info=groups[g], cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
}
```





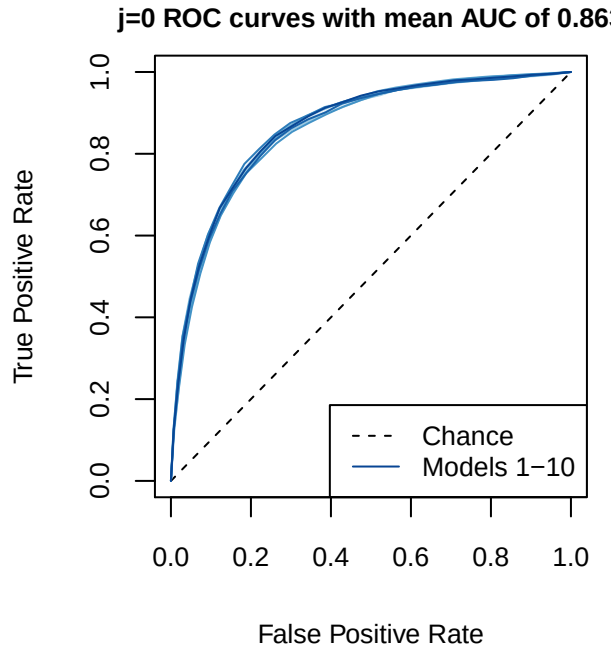
Overall the models perform pretty similarly in terms of AUC, though the j=2 group has performed slightly worse. Interestingly the AMS graphs look quite different and the optimal decision threshold is different for each group. For the j=2+ group the AMS score is extremely noisy after t=0.7.

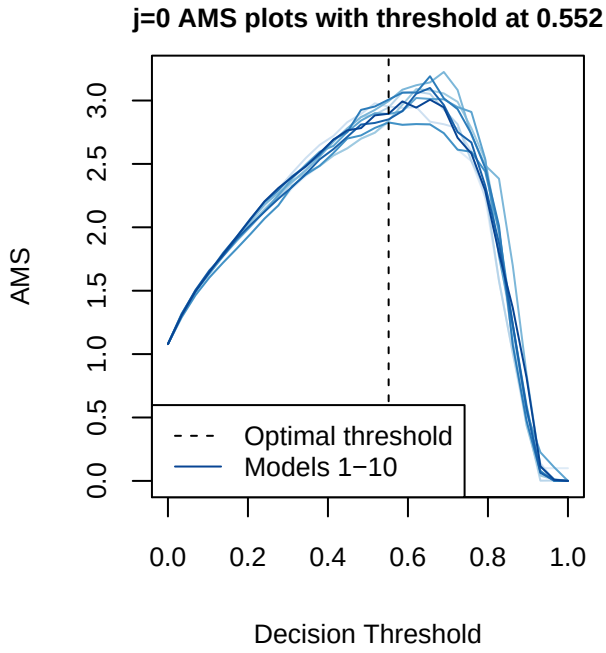
Standardising data

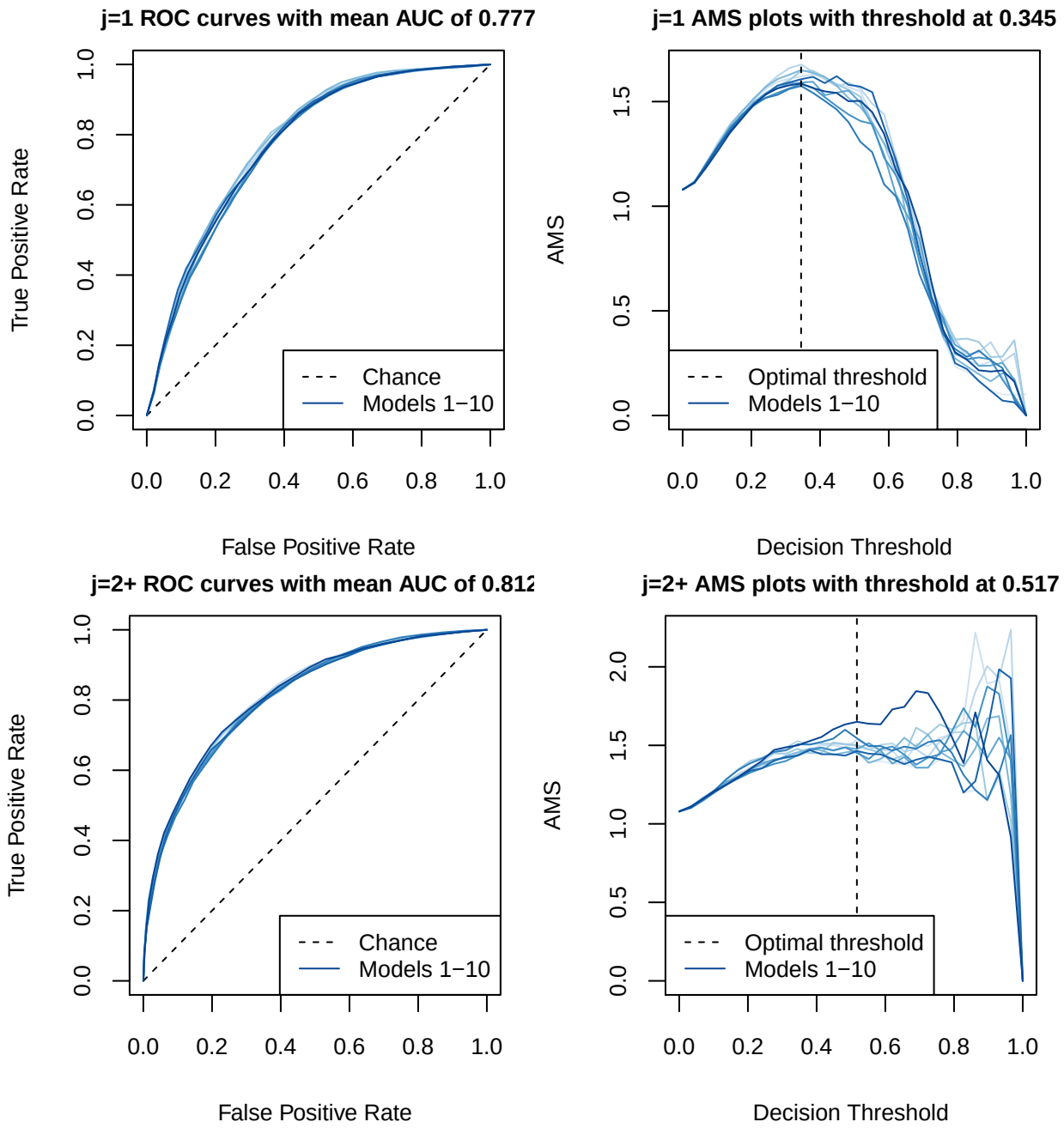
It is typically beneficial in regression problems to standardise the data before training. That is, to centre each variable to have a mean of 0 and standard deviation of 1. Here we'll see how replacing any remaining -999s with NAs (this will just effect the `DER_mass_MMC` column) and standardising the variables using our `scale_dat` function affects performance.

```
par(mfrow=c(1,2))
par(mar=c(4,4,2,1))

for(g in 1:G){
  X_group <- X[info$Group==groups[g], !colnames(X) %in% features_to_rm[[g]]]
  info_group <- info[info$Group==groups[g],]

  # get an index for CV groups
  k <- 10
  kI <- partition_data(n=nrow(X_group), k=k, random=T)

  #create lists to hold the k models and k roc curves
  models <- vector("list", k)
  rocs <- vector("list", k)
  amss <- vector("list", k)

  for(i in 1:k){
    X_train <- X_group[kI != i,]
    y_train <- info_group[kI != i, "Y"]

    X_test <- X_group[kI == i,]
    y_test <- info_group[kI == i, "Y"]
    w_test <- info_group[kI == i, "Weight"]

    #scale the training data, and scale the test data with the same transformation
```

```
    X_train_scaled <- scale_dat(X_train, X_train)
    X_test_scaled <- scale_dat(X_test, X_train)

    #fit a logistic regression model to the CV training data
    model <- logistic_model$new(X=X_train_scaled, y=y_train)

    #use it to predict the classifications of the test data
    prob <- model$predict(X_test_scaled)

    #store roc and ams data
    rocs[[i]] <- ROC_curve$new(y_test, prob)
    amss[[i]] <- AMS_data$new(y_test, prob, w_test)
  }

  plot_rocs(rocs, info=groups[g], cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
  plot_amss(amss, info=groups[g], cex.main=0.8, cex.lab=0.8, cex.axis=0.8)
}
```







We can see that standardising the data has improved performance for all three groups, though the j=2+ group still has noisy AMS. So far we have just used a standard logistic regression which finds a linear boundary and so performance is likely to improve when we add some non-linearity to the models.