Feature selection

Anthony Stephenson

1/3/2021

Logistic Regression and Feature Engineering

In an effort to include higher order features we decided to implement feature augmentation in the form of polynomial transformations and RBF centroid features. We created a method in which we could vary the features easily with a few parameters so that we could run a series of experiments, with 10-fold cross-validation. Then we can compare the performance of the models to obtain a "best" model to apply to our hold-out validation set (the private leaderboard set "v").

From the exploratory data analysis we saw that the data can be viewed as having been generated from different underlying processes, based on the structure of the missing data. Although we considered building independent models for all 6 of the missing data patterns, we found that some of the groups were too small to build reliable models. As a result we opted to divide only by jet number (rather than Higgs mass as well) and remove the remaining missing columns as required.

A set of functions (project_funcs.r) are included in the `lhc` package to streamline the pipeline for the model experiments. Example functions are `import_data` to load and pre-process the Kaggle dataset, and `idx_jet_cat` which indexes samples from the jet number groups.

Initialise script

Set script parameters:

```
set.seed(22)

# Regularisation (L2) parameter [global]
lambda <- 1e-4
# constraint parameter for L1 Logistic regression [global]
C <- 1
# number of jet/Higgs mass groups to build different models for
G <- 3
poly_order <- 2
n_rbf <- 0

# AMS thresholds
thresholds <- c(0.6, 0.4, 0.6)

# choose model. Interchangeable so long as we have implemented polymorphism across the model classes so
# L2 logistic regression
model_init <- partial(logistic_model$new, lambda=lambda)

# Constrained logistic regression (using CVXR)
# model_init <- partial(logistic_l1_model$new, C=C)
```

```r
# pick training and validation sets
train_label = c("t")
val_label <- c("v")

# name to use in the .csv output of results
result_label <- "test_private_leaderboard"

# bool to choose whether to save outputs
if (!("do_save_outputs" %in% ls())) {
  do_save_outputs <- TRUE
}
# dir to save figures
fig_dir <- path_join(c(dirname(getwd()), "doc/figs/"))
```

Load data

```r
filepath <- list.files(path="~", pattern="atlas-higgs-challenge-2014-v2.csv",
                       full.names=T, recursive=T)
data <- import_data(filepath)
```

Assign variables corresponding to the training set.

```r
train_idx <- get_subset_idx(data$kaggle_s, train_label)
X <- data$X[train_idx, ]
y <- data$y[train_idx]
kaggle_w <- data$kaggle_w[train_idx]
w <- data$kaggle_w[train_idx]
nj <- data$nj[train_idx]
e_id <- data$e_id[train_idx]
```

Assign variables corresponding to the validation set.

```r
# public leaderboard set
val_idx <- get_subset_idx(data$kaggle_s, val_label)
Xv <- data$X[val_idx, ]
yv<- data$y[val_idx]
wv <- data$kaggle_w[val_idx]
njv <- data$nj[val_idx]
```

Feature engineering

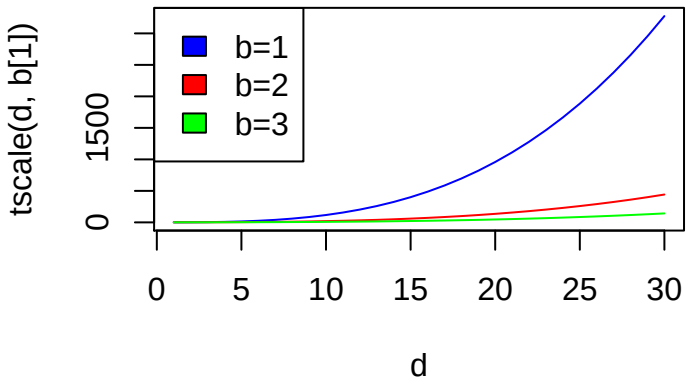
Apply some feature engineering

```r
# modify features
X <- reduce_features(X)
X <- invert_angle_sign(X)
Xv <- reduce_features(Xv)
Xv <- invert_angle_sign(Xv)

if (poly_order > 1) {
  X <- poly_transform(X, poly_order)
  Xv <- poly_transform(Xv, poly_order)
}

# ensure X and Xv have the same columns
```

```
cols2keep <- intersect(colnames(X), colnames(Xv))
Xv <- Xv[, cols2keep]
X <- X[, cols2keep]

#Xv <- scale_dat(Xv, X, na.rm=TRUE)
```

Should we include interaction terms? Plot computation scaling as a function of number of initial features d and order of polynomial terms of those to include, if we were to include all pairwise interactions.

```
nint <- function(d) {
  exp(lgamma(d+1) - log(2) - lgamma(d-1))
}
tscale <- function(d,b) {
  dI <- nint(d)
  1 + dI^3/(b*d)^3 + dI^2/(b*d)^2 + dI/(b*d)
}
d <- 1:30
b <- 1:3
plot(d, tscale(d,b[1]), type="l", col="blue")#, log="y")
lines(d, tscale(d,b[2]), col="red")
lines(d, tscale(d,b[3]), col="green")
legend("topleft", legend=c("b=1", "b=2", "b=3"), fill=c("blue", "red", "green"))
```



Compute some initial parameters

```
s <- avg_median_pairwise_distance(X)

# K-Fold CV partitioning
K <- 10
kI <- partition_data(length(y), K, random = TRUE)

# number of rows and columns of data matrix
n <- nrow(X)
d <- ncol(X) + n_rbf

# sum weights for to renormalise for AMS in partitions
#sum_w <- sum(w)

# set colours for jet groups
colours <- generate_colours(G)

#find the jet groups
```

Use missing data pattern to partition data.

```
# get missing rows. separate by number of jets and presence of Higgs mass and fit separate models
# find columns with features with any missing values for each number of jets: 0, 1, 2+ in combination w
jet_cats <- c(1:3, 1:3)
features_to_rm <- set_features_to_rm(X, G, kI, nj)
```

Run CV experiments

Loop over jet groups and folds and fit a model for each (so GK total) and then test each of these on their OOS subset, recording AUC and AMS.

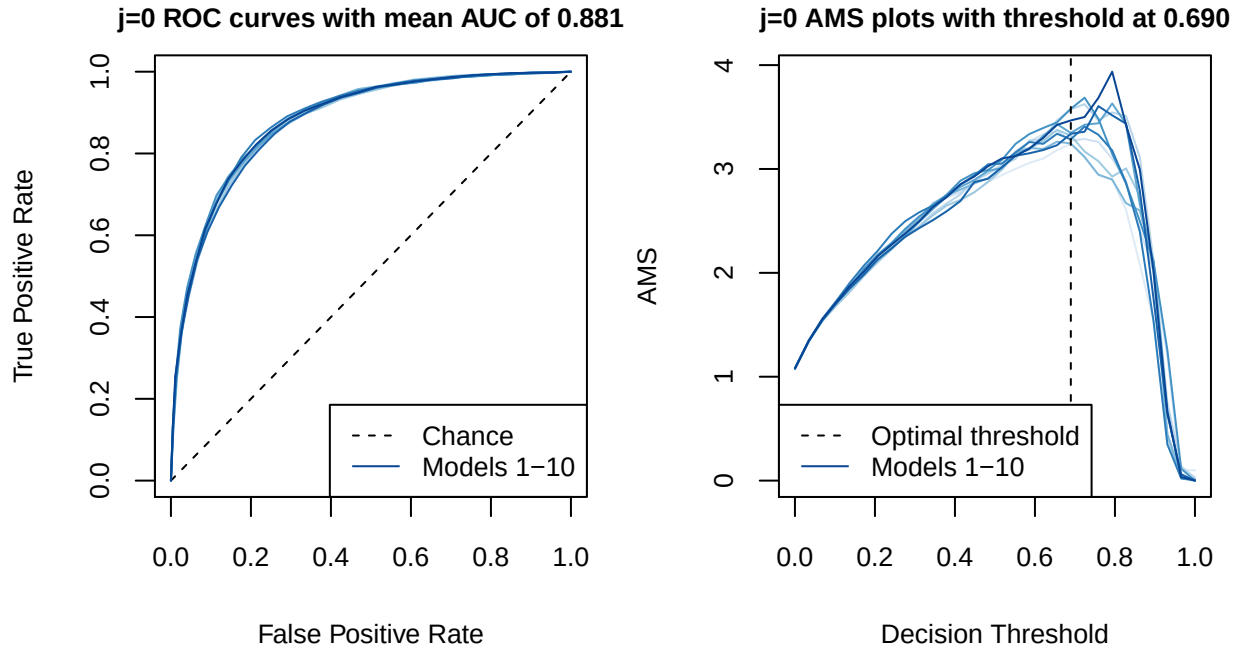
```
# loop over folds
#create lists to hold the k models and k roc curves
models <- vector("list", G*K)
rocs <- vector("list", G*K)
ams_obj <- vector("list", G*K)
b <- matrix(NA, nrow=d, ncol=G*K)

ams <- rep(NA, length=G*K)
auc <- rep(NA, length=G*K)

par(mfrow=c(2, 3))
for (mj in 1:G) {
    # loop over sets of jet number {0, 1, 2+} and mH presence/absence
    for (k in 1:K) {
        j <- jet_cats[mj]
        # get train and test split indices for this jet category and fold
        fit_row_idx <- kI != k & idx_jet_cat(nj, j)
        test_row_idx <- kI == k & idx_jet_cat(nj, j)

        # add r RBF centroid features, using the same reference centroids in training and testing sets
        if (n_rbf > 0) {
          rbf_centroids <- get_rbf_centroids(X[fit_row_idx, ], n_rbf)
          Xi <- rbf_centroids$"xi"
          Xtrain <- add_rbf_features(X[fit_row_idx, ], s, n_rbf, Xi=Xi)
          Xtest <- add_rbf_features(X[test_row_idx, ], s, n_rbf, Xi=Xi)
        } else {
          Xtrain <- X[fit_row_idx, ]
          Xtest <- X[test_row_idx, ]
        }

        # get indices for columns to include for this jet category
        col_idx <- get_valid_cols(colnames(X), features_to_rm, mj)

        # define train and test matrices
        Xtrain <- as.matrix(Xtrain[, col_idx])
        Xtest <- as.matrix(Xtest[, col_idx])

        # standardize the data (using training as a reference for the test set to avoid using any test
        Xtest <- scale_dat(Xtest, Xtrain)
        Xtrain <- scale_dat(Xtrain, Xtrain)

        # get index of this model, from jet category and fold (to retrieve in results arrays)
        model_idx <- get_model_idx(mj, k, K)
```

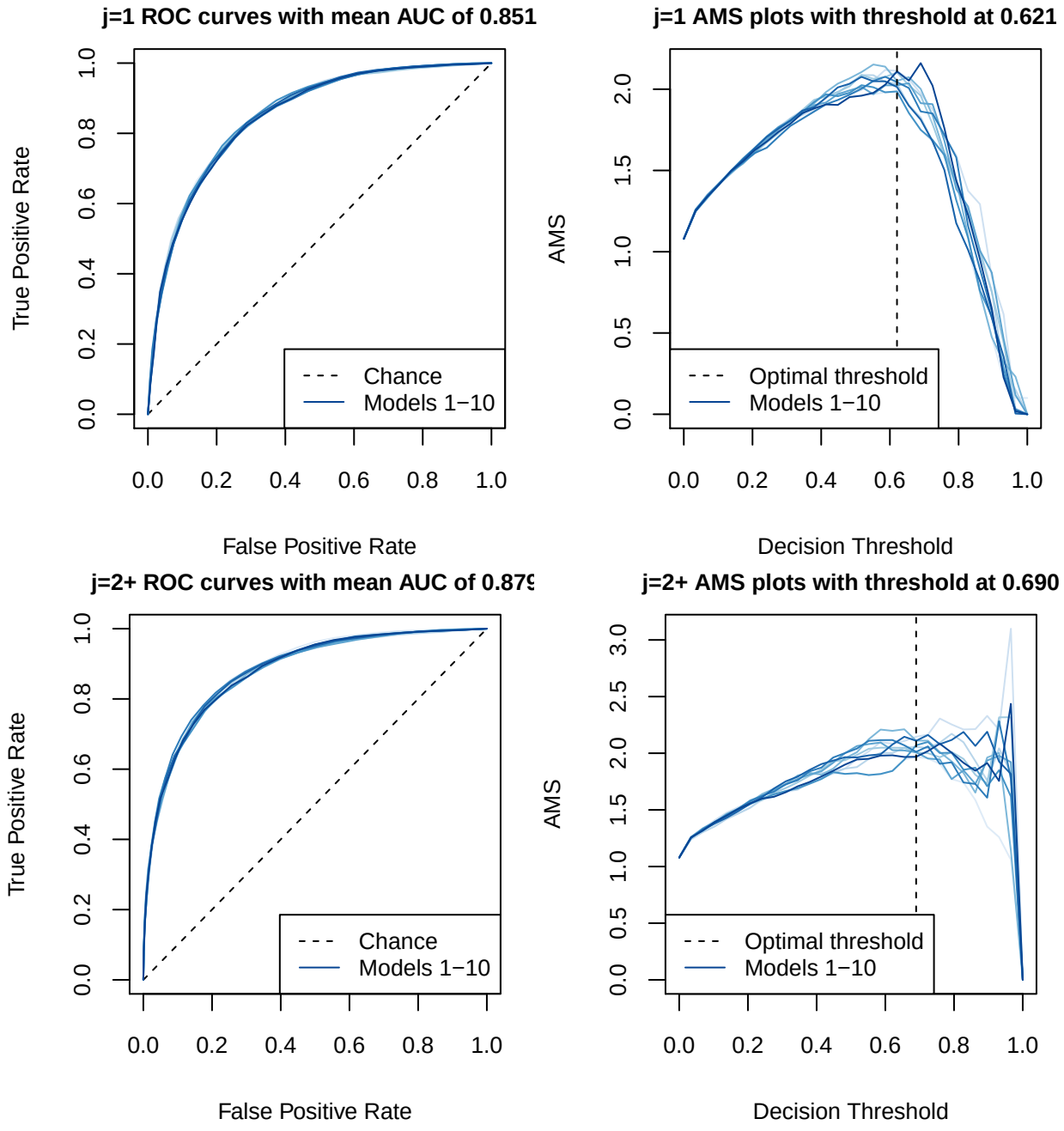
```
        # fit a logistic regression model to the CV training data
        models[[model_idx]] <- model_init(X=Xtrain, y=y[fit_row_idx])

        # use it to predict the classifications of the test data
        p_hat <- models[[model_idx]]$predict(Xtest)

        # create an ROC curve object
        rocs[[model_idx]] <- ROC_curve$new(y[test_row_idx], p_hat)

        # create an AMS curve object
        w_this_partition <- w[test_row_idx] #* sum_w/sum(w[test_row_idx])
        ams_obj[[model_idx]] <- AMS_data$new(y[test_row_idx], p_hat, w_this_partition)#, sum_w=sum_w)
    }
}
```

Plot CV metrics

Plot ROC curves and AMS against decision threshold over the k folds for each group. By plotting AMS against threshold we can get a feeling for what threshold we should use for our final model and how consistent they are across folds. To account for the noise (especially important for the group j=2+) we find the minimum curve over the K fold curves (by taking the minimum over folds at each threshold point) and then find the threshold which maximises this curve.

```
par(mfrow=c(1,2))
par(mar=c(4,4,2,1))
for(j in 1:G){
  i1 <- get_model_idx(j, 1, K)
  i2 <- get_model_idx(j, K, K)

  plot_rocs(rocs[i1:i2], info=groups[j])
  plot_amss(ams_obj[i1:i2], info=groups[j])
}
```



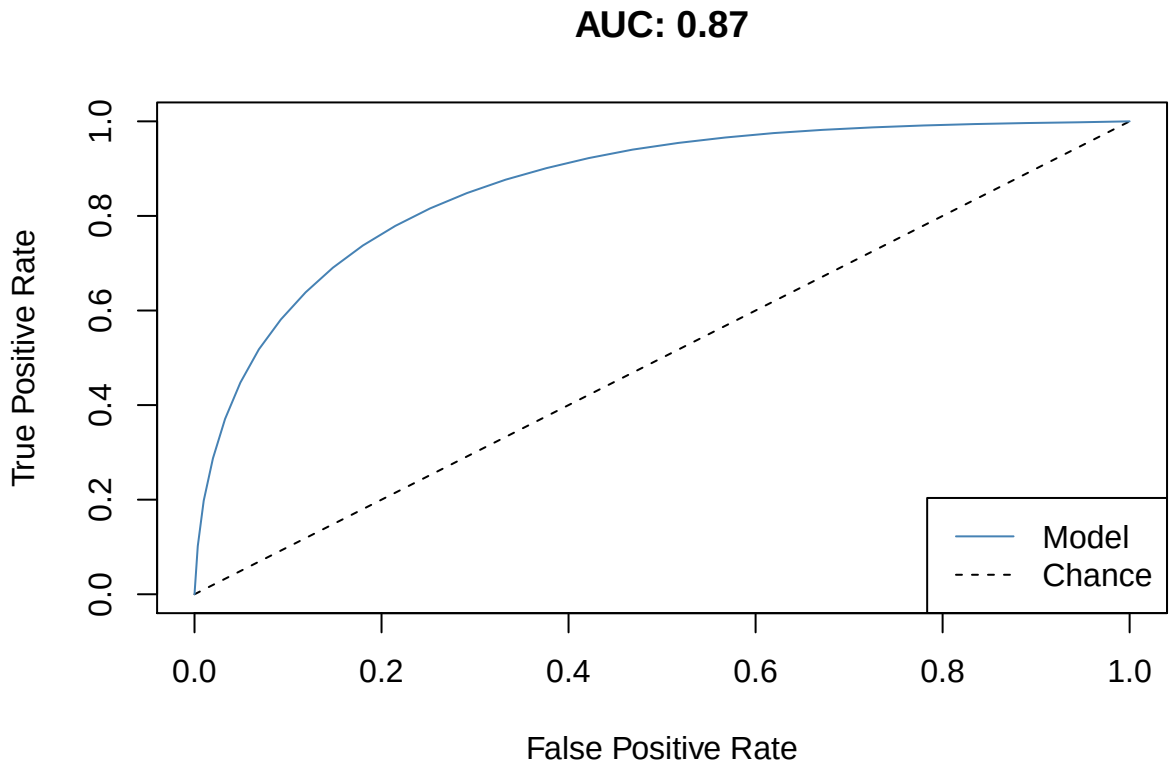


Run full model

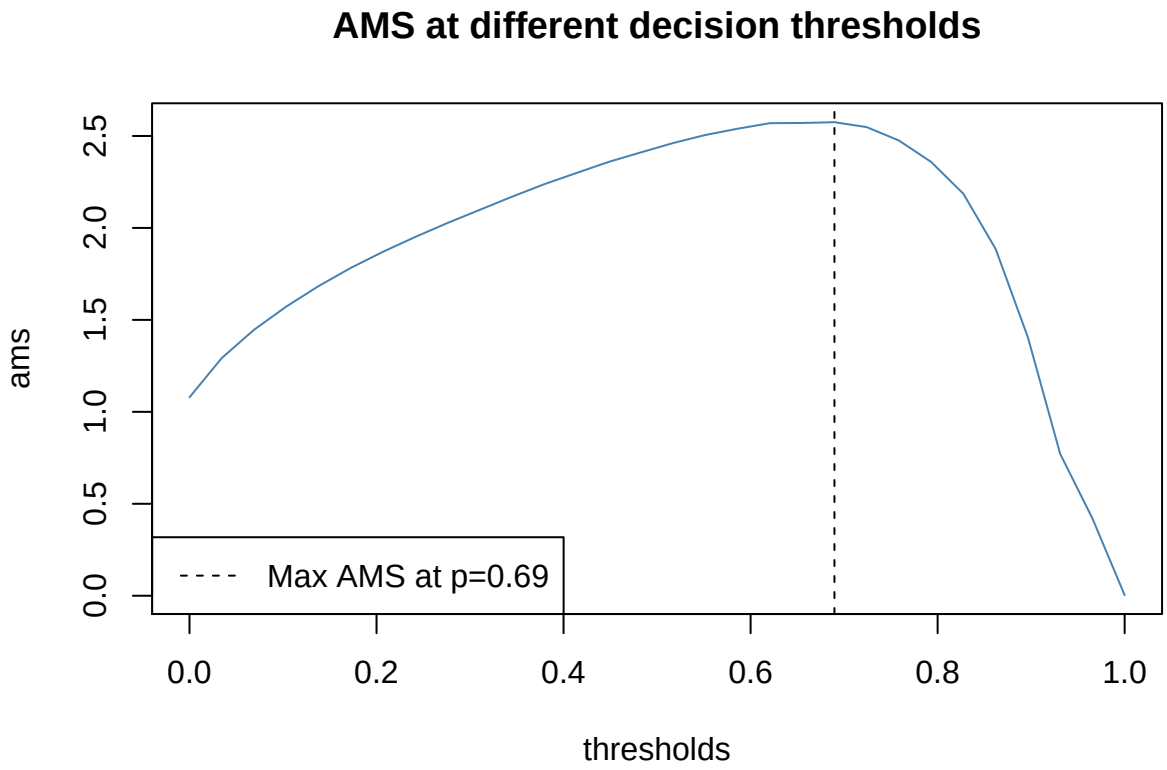
Fit model (for each category) on whole (unfolded) dataset

```
predictions <- rep(NA, length(Xv))
rbf_idx <- matrix(NA, nrow=G, ncol=n_rbf)

for (mj in 1:G) {
    # loop over sets of jet number {0, 1, 2+} and mH presence/absence
    j <- jet_cats[mj]
    fit_row_idx <- idx_jet_cat(nj, j) #& idx_higgs_mass(X, mj, G)

    if (n_rbf > 0) {
```

```
      # add r RBF centroid features, using the same reference centroids in training and testing sets
      rbf_centroids <- get_rbf_centroids(X[fit_row_idx, ], n_rbf)
      Xi <- rbf_centroids$"xi"
      # record which rows to use for OOS in public set (or any other OOS)
      rbf_idx[mj, ] <- rbf_centroids$"idx"
      Xtrain <- add_rbf_features(X[fit_row_idx, ], s, n_rbf, Xi=Xi)
    } else {
      Xtrain <- X[fit_row_idx, ]
    }

    test_row_idx <- idx_jet_cat(njv, j)# & idx_higgs_mass(Xv, mj, G)
    if (n_rbf > 0) {
      # add r RBF centroid features, using the same reference centroids in training and testing sets
      Xi <- X[rbf_idx[mj, ], ]
      Xtest <- add_rbf_features(Xv[test_row_idx, ], s, n_rbf, Xi=Xi)
    } else {
      Xtest <- Xv[test_row_idx, ]
    }

    # get indices for columns to include for this jet category
    col_idx <- get_valid_cols(colnames(Xtrain), features_to_rm, mj)
    # standardize the data
    Xtrain <- as.matrix(Xtrain[, col_idx])
    Xtest <- as.matrix(Xtest[, col_idx])
    # Xtest <- scale_dat(Xtest, Xtrain)
    # Xtrain <- scale_dat(Xtrain, Xtrain)

    #fit a logistic regression model to the all of the training data
    model <- model_init(X=Xtrain, y=y[fit_row_idx])

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

    predictions[test_row_idx] <- p_hat
}

#create objects for overall results
full_roc <- ROC_curve$new(yv, predictions)
amsv_obj <- AMS_data$new(yv, predictions, wv)

full_roc$plot_curve()
```




```
amsv_obj$plot_ams()
```




```
#if we use the threshold of 0.6 for each group we get an overall AMS of
y_pred <- predictions >= 0.6
ams_metric(yv, y_pred, wv)
```

```
## [1] 2.548925
```