Identifying fermionic decay signals of the Higgs boson with classification algorithms

Georgie Mansell, Anthony Stephenson

25/1/2021

Contents

{r child = '1_DataExploration.rmd'}

Load packages

Introduction

We decided that running a logistic regression model would be a good, robust model to use as, at the very least, a benchmark. In an effort to include higher order features than the original dataset, we decided to implement feature augmentation in the form of polynomial transformations and RBF centroid features that we could apply easily with the use of a few parameters in the script. The hope was that we could run a series of experiments exploring the parameter space, with 10-fold cross-validation, and analyse the results to obtain a "best" model to apply to a hold-out test set.

The Kaggle data is divided into various subsections, each labelled with a character: Kaggle sets: "t":training, "b":public leaderboard, "v":private leaderboard, "u":unused.

"t" is used for the cross-validation procedure and as a final full fitting set before testing on the "v" hold-out set. The "b" set is relatively small, so has been excluded for simplicity.

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 each of these groups, 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. This appeared to provide more robust and better performance.

Initialise script

Set script parameters:

```
# 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 <- 3

# 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)

# 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

Load the data

```
# divide data into training kaggle set, and retain hold-out (before further cross-validation partitioni
source_path <- path_join(c(path_dir(path_dir(getwd()))))
filename <- "atlas-higgs-challenge-2014-v2.csv"
filepath <- path_join(c(source_path, "LHC_dump", "R", filename))
data <- import_data(filepath)
```

Assign variables corresponding to the training set.

```
train_idx <- get_subset_idx(data$kaggle_s, train_label)
X <- data$X[train_idx, ]
y <- data$y[train_idx]
# need to use kaggle weignts to make AMS correct?!
kaggle_w <- data$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.

```
# 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

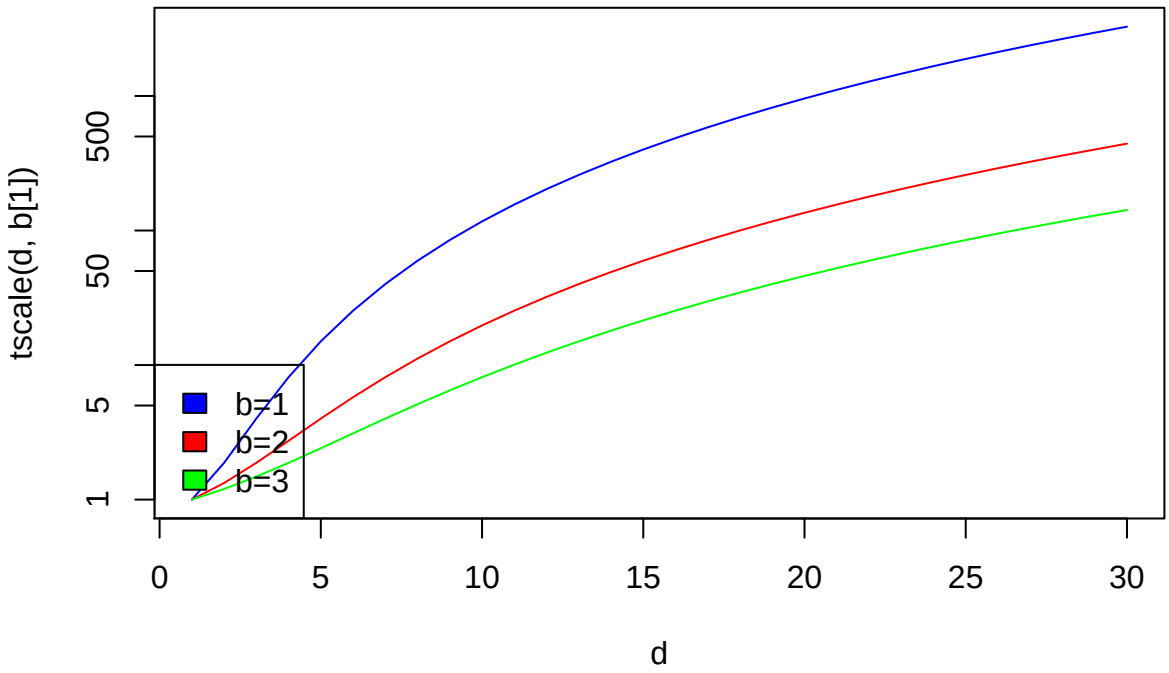
```
# modify features
X <- reduce_features(X)
X <- invert_angle_sign(X)
Xv <- reduce_features(Xv)
Xv <- invert_angle_sign(Xv)

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="lin", col="blue", log="y")
```

```
## Warning in plot.xy(xy, type, ...): plot type 'lin' will be truncated to first
## character
```

```
lines(d, tscale(d,b[2]), col="red")
lines(d, tscale(d,b[3]), col="green")
legend("bottomleft", 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)
```

```
w_ratio <- function(y, w, nj, label, j) {
  sum(w[idx_jet_cat(nj, j) & y==label])/sum(w[idx_jet_cat(nj, j)])
}
weight_stats <- data.frame(j=c(1,2,3),
                           s=c(w_ratio(y,w,nj,1,1), w_ratio(y,w,nj,1,2), w_ratio(y,w,nj,1,3)),
                           b=c(w_ratio(y,w,nj,0,1), w_ratio(y,w,nj,0,2), w_ratio(y,w,nj,0,3))
)
weight_stats
n_ratio <- function(y, nj, label, j) {
  sum(y==label & idx_jet_cat(nj, j))/sum(idx_jet_cat(nj, j))
}
n_stats <- data.frame(j=c(1,2,3),
                      s=c(n_ratio(y,nj,1,1), n_ratio(y,nj,1,2), n_ratio(y,nj,1,3)),
                      b=c(n_ratio(y,nj,0,1), n_ratio(y,nj,0,2), n_ratio(y,nj,0,3))
)
```

```
n_stats
```

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)
```

```
n_rows_p_partition <- matrix(, nrow=G*K, ncol=2)
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]
        fit_row_idx <- kI != k & idx_jet_cat(nj, j) & idx_higgs_mass(X, mj, G)
        test_row_idx <- kI == k & idx_jet_cat(nj, j) & idx_higgs_mass(X, mj, G)

        model_idx <- get_model_idx(mj, k, K)

        n_rows_p_partition[model_idx, 1] <- sum(fit_row_idx)
        n_rows_p_partition[model_idx, 2] <- sum(test_row_idx)
    }
}
n_rows_p_partition
```

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.

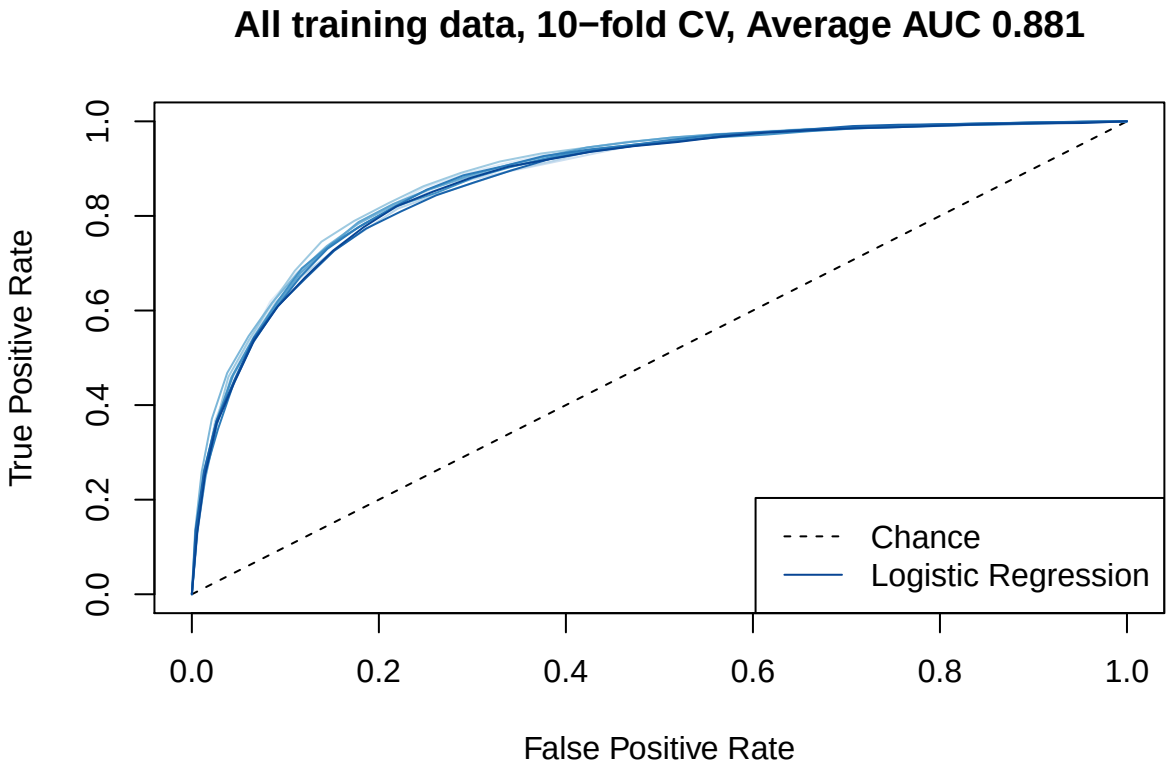
```
# loop over folds
#create lists to hold the k models and k roc curves
models <- vector("list", G*K)
rocs <- vector("list", G*K)
ams_obj <- vector("list", G*K)
b <- matrix(, nrow=d, ncol=G*K)

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/or higgs mass) and fold
        fit_row_idx <- kI != k & idx_jet_cat(nj, j) & idx_higgs_mass(X, mj, G)
        test_row_idx <- kI == k & idx_jet_cat(nj, j) & idx_higgs_mass(X, mj, G)

        # 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)
```

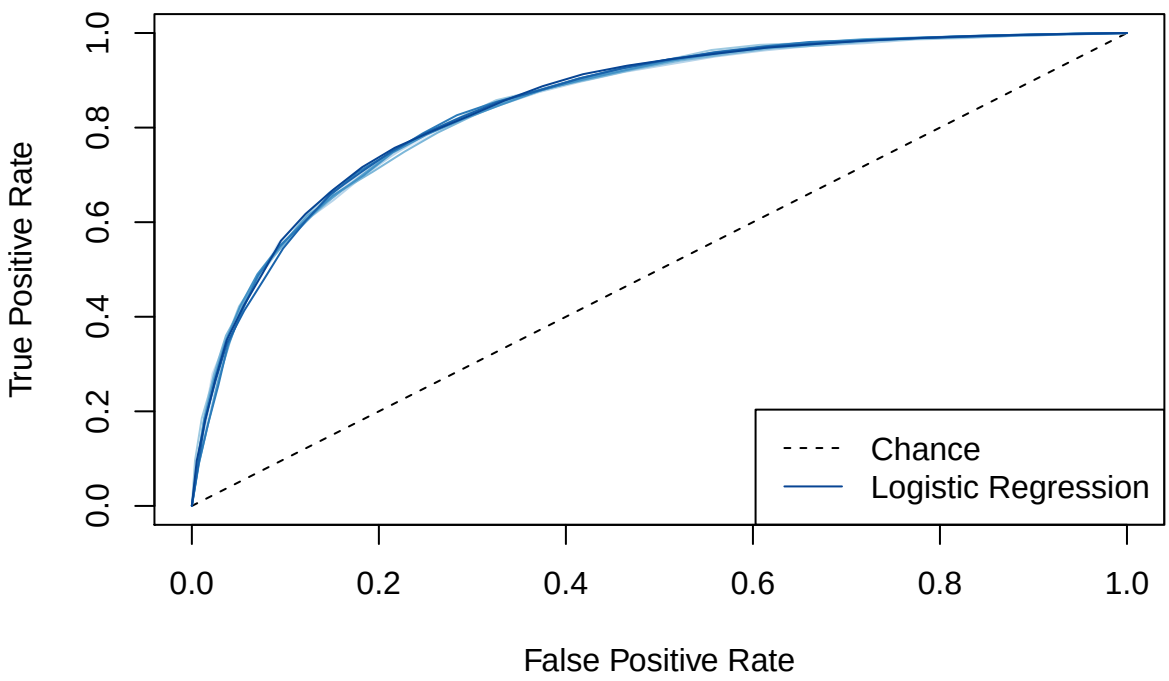
```
        } else {
          Xtrain <- X[fit_row_idx, ]
          Xtest <- X[test_row_idx, ]
        }

        # 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 matricies
        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)
    }
}
```

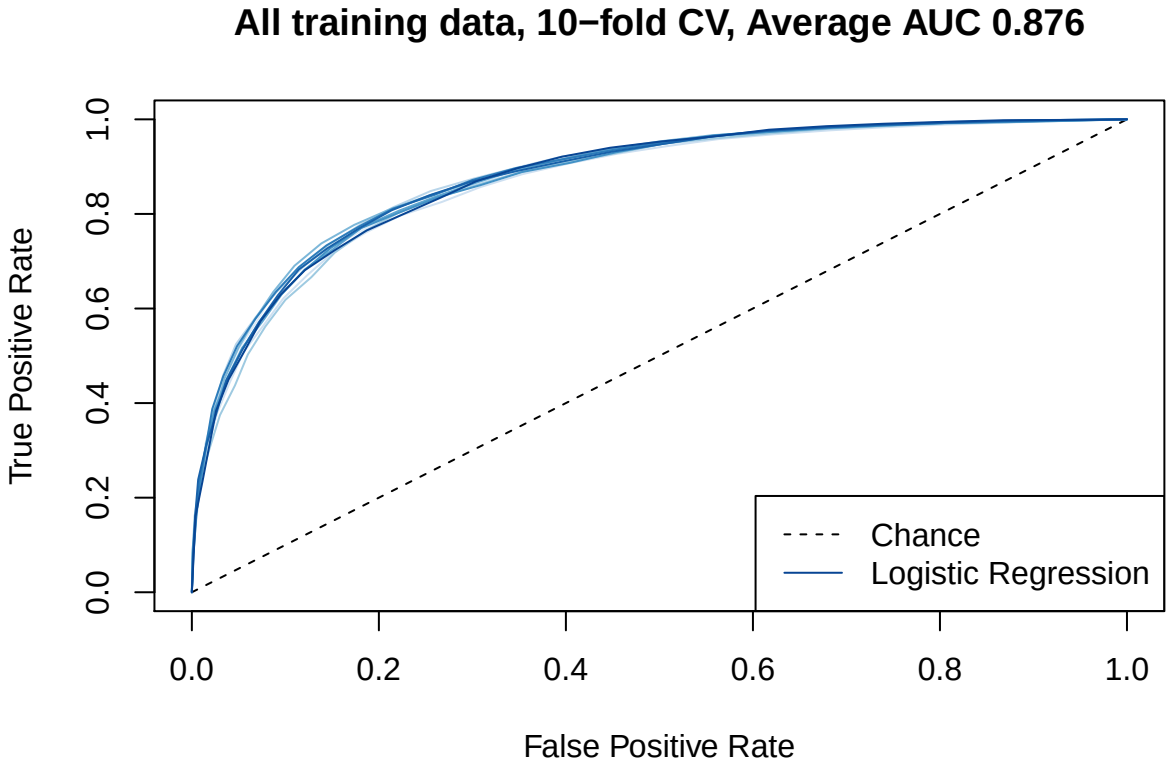
Plot CV metrics

Plot ROC curves for each fold for each group and calculate mean AUC over folds for each group.

```
auc <- sapply(rocs, function(x) x$calc_auc())
for(j in 1:G){
  #find average auc over folds
  i1 <- get_model_idx(j, 1, K)
  i2 <- get_model_idx(j, K, K)
  average_auc <- mean(sapply(rocs[i1:i2], function(x) x$auc), na.rm=TRUE)
  #get a plot of the k ROC curves on the same axis
  if (do_save_outputs) {
    save_fig(partial(plot_rocs, rocs[i1:i2], title=sprintf("All training data, %i-fold CV, Average AUC %
             path_join(c(fig_dir, sprintf("cvoos-auc-curves%i.pdf", j))))
  }
  # pdf(file=path_join(c(fig_dir, sprintf("cvoos-auc-curves%i.pdf", j))))
  plot_rocs(rocs[i1:i2], title=sprintf("All training data, %i-fold CV, Average AUC %.3f", K, round(avera
  # dev.off()
}
```

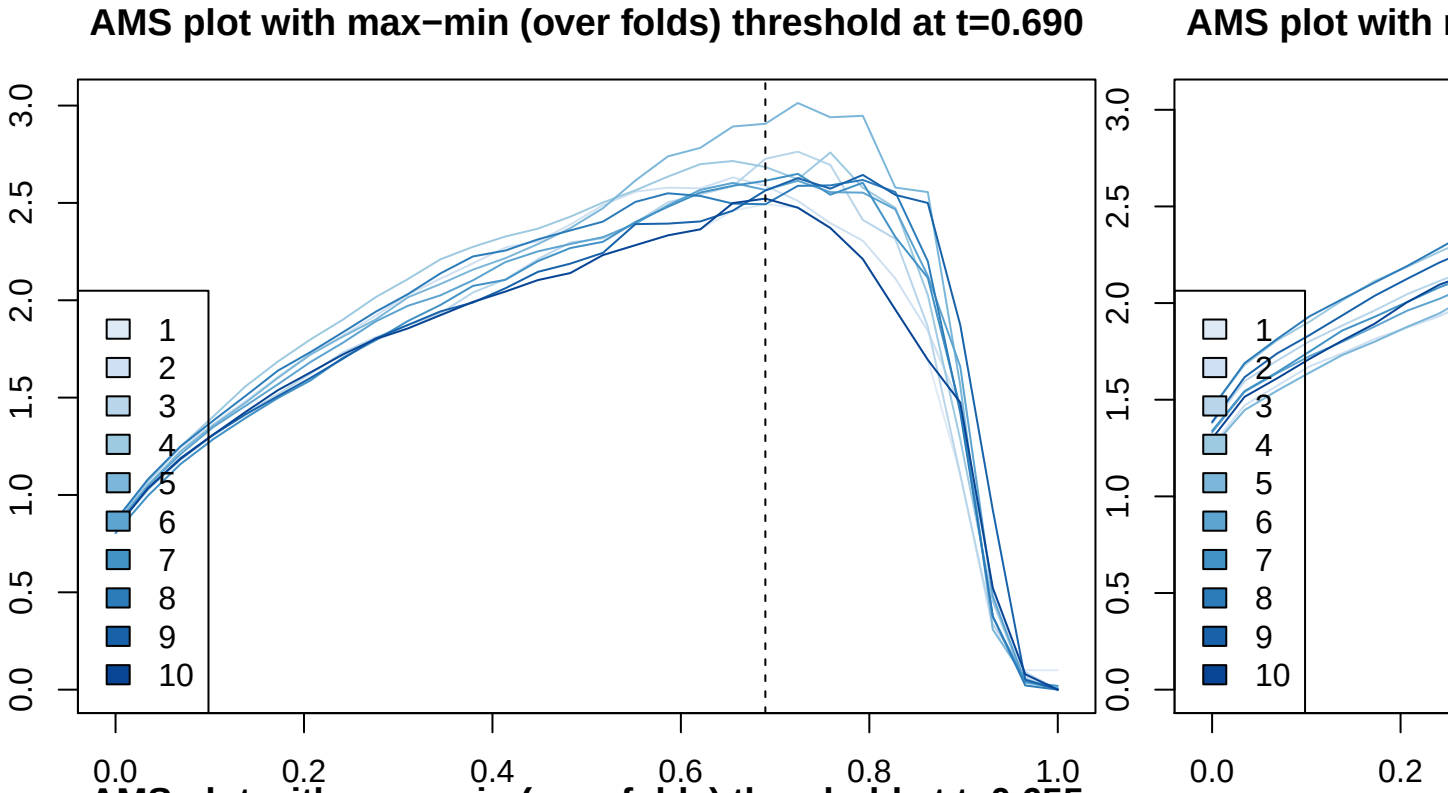

All training data, 10−fold CV, Average AUC 0.881
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Chance
Logistic Regression

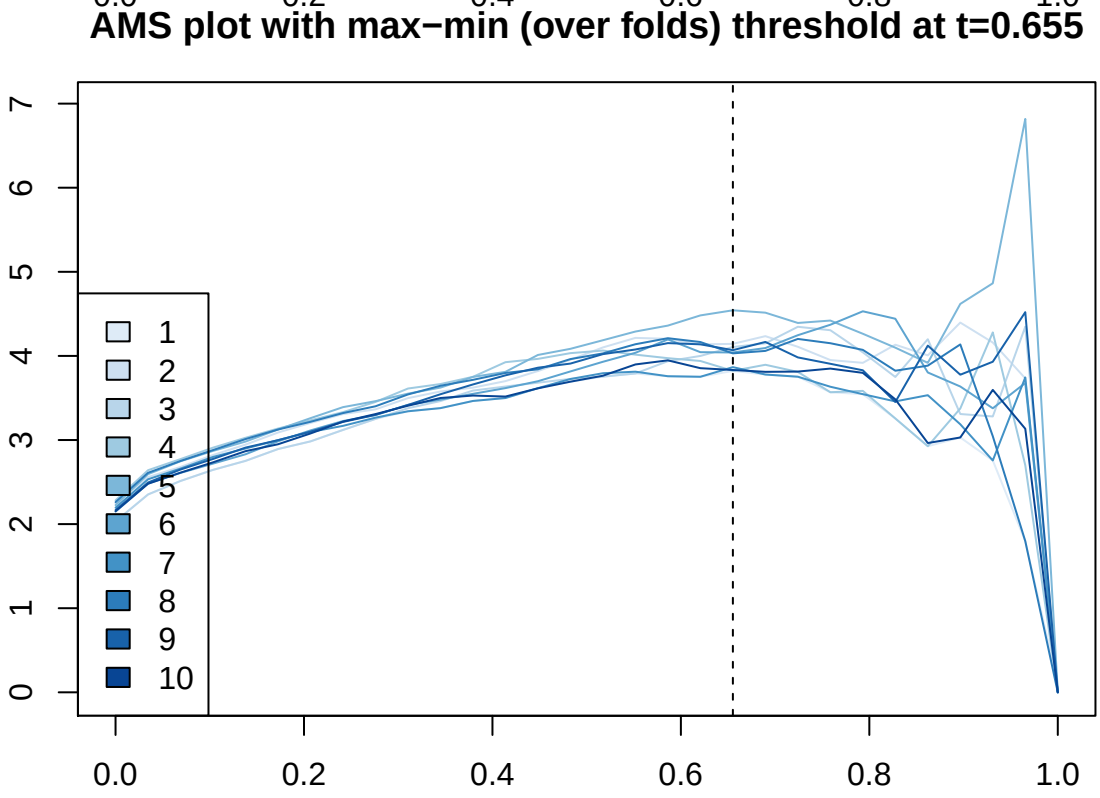

All training data, 10−fold CV, Average AUC 0.849
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Chance
Logistic Regression



If we now also plot curves of the AMS against threshold we can get a feeling for what we should choose and how consistent they are across folds (for each model type). They are all relatively noisy, although `j=3` is comfortably the worst of the three in this respect. In an effort to account for this, and reduce the chance of overfitting the thresholds, we have taken to calculating the minimum curve over the K fold curves (by taking the minimum over folds at each threshold point) and then finding the maximum threshold for this curve. The results seem more robust, and give approximately $(0.6, 0.4, 0.6)$.

```
ams <- sapply(ams_obj, function(x) x$calc_ams())
par(mar=c(3,3,3,3))
for(j in 1:G){
  #find average auc over folds
  i1 <- get_model_idx(j, 1, K)
  i2 <- get_model_idx(j, K, K)
  # calculate the minimum AMS over the curves at each point to get an overall lower bound curve
  #get a plot of the k ROC curves on the same axis
  if (do_save_outputs) {
    save_fig(partial(plot_amss, ams_obj[i1:i2]),
             path_join(c(fig_dir, sprintf("cvoos-ams-curves%i.pdf", j))))
  }
  # pdf(file=path_join(c(fig_dir, sprintf("cvoos-ams-curves%i.pdf", j))))
  plot_amss(ams_obj[i1:i2])
  # dev.off()
}
```





Run full model

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

```
# loop over folds
#create lists to hold the k models and k roc curves
full_models <- vector("list", G)
full_rocs <- vector("list", G)
rbf_idx <- matrix(, nrow=G, ncol=n_rbf)
# check warnings?
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, ]
    }

    # 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])
    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, 1, 1)

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

Test on (until now unseen) validation set. Calling it validation as we can then use the summary metrics to check free parameters, being careful not to overdo it.

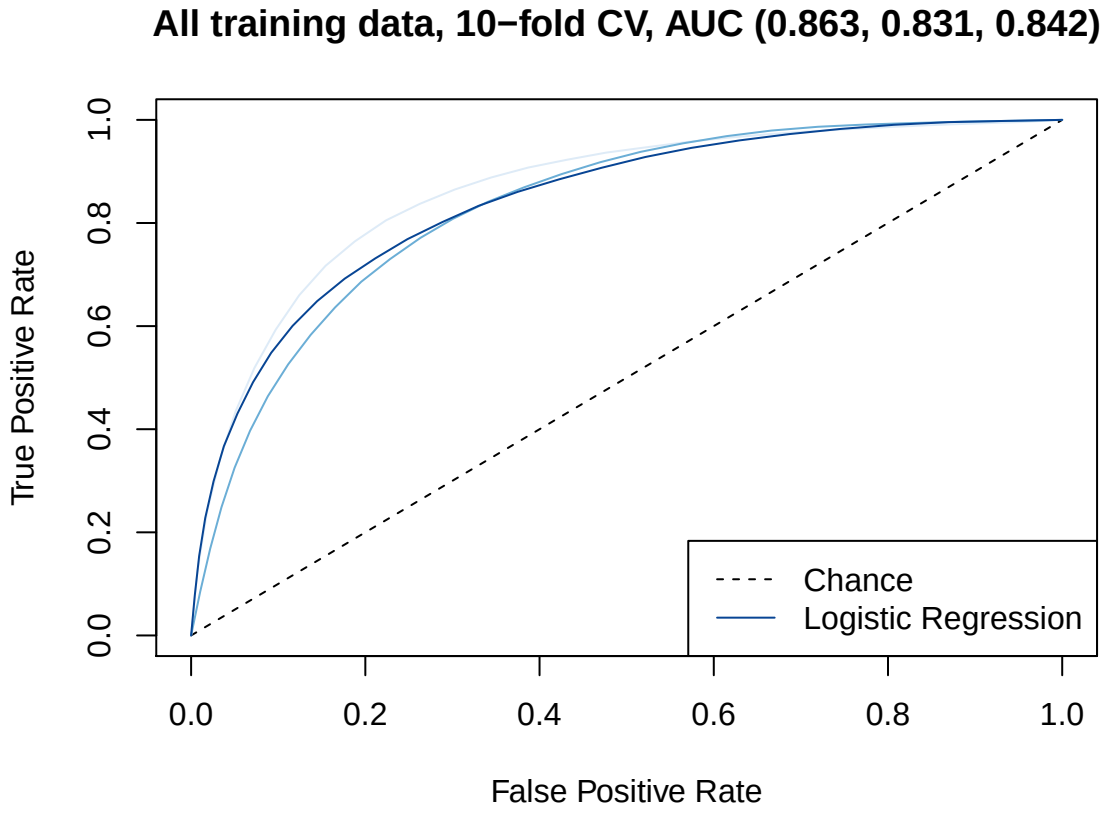
```
amsv_obj <- vector("list", G)

# check warnings
sum_wv <- sum(wv)
for (mj in 1:G) {
    j <- jet_cats[mj]
    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, ]
    }
```

```
    col_idx <- get_valid_cols(colnames(Xtest), features_to_rm, mj)

    Xtest <- as.matrix(Xtest[, col_idx])

    model_idx <- get_model_idx(mj, 1, 1)

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

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

    w_this_partition <- wv[test_row_idx] * sum_wv/sum(wv[test_row_idx])
    amsv_obj[[model_idx]] <- AMS_data$new(yv[test_row_idx], p_hat, w_this_partition)#, sum_w=sum_wv)
}
```

Plot test metrics

Plot ROC curves for each group on the validation set, with AUC for each.

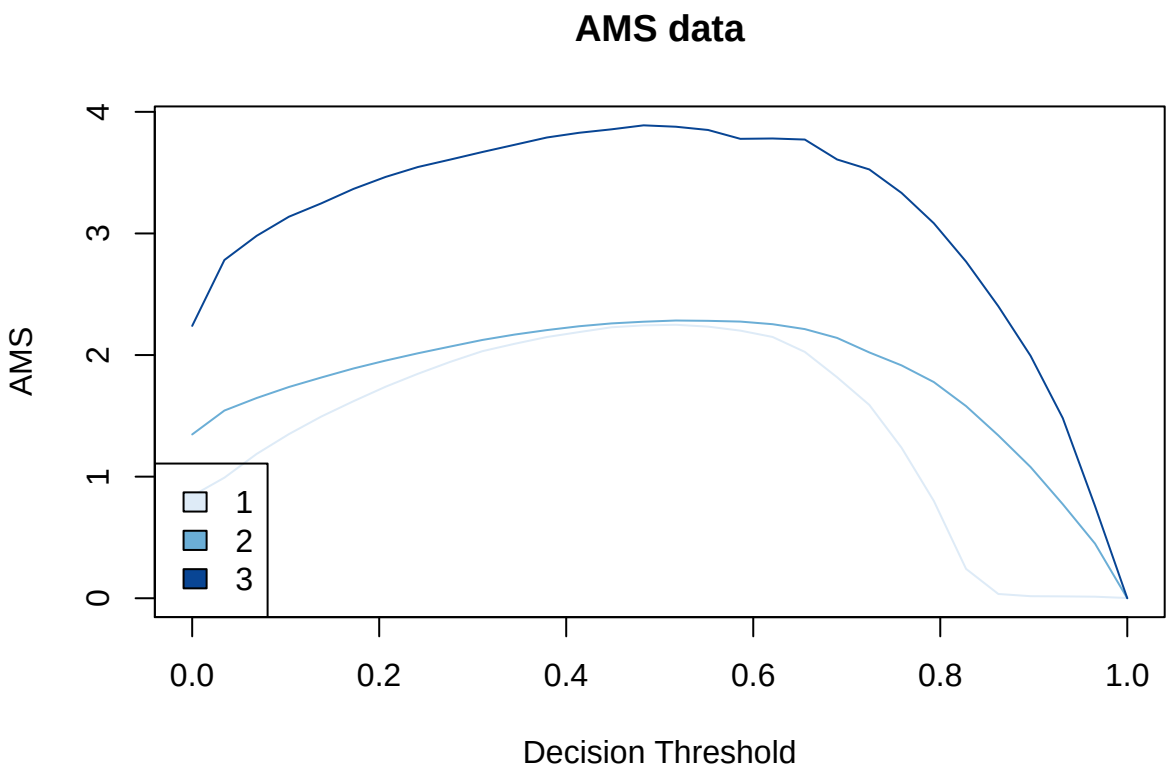
```
. <- sapply(full_rocs, function(x) x$calc_auc())
aucv <- sapply(full_rocs, function(x) round(x$auc, 3))
# pdf(file=path_join(c(fig_dir, "validation-auc-curves.pdf")))
par(mar=c(4,4,4,4))
if (do_save_outputs) {
  save_fig(partial(plot_rocs, full_rocs, title=sprintf("All training data, %i-fold CV, AUC (%.3f, %.3f,
           path_join(c(fig_dir, "validation-auc-curves.pdf")))
}
plot_rocs(full_rocs, title=sprintf("All training data, %i-fold CV, AUC (%.3f, %.3f, %.3f)", K, round(au
```




```
# dev.off()
```

Now plotting the AMS thresholds for the validation set, we see that they differ substantially from those we saw in the training set, despite the K-fold CV. The validation set is much smaller than the full training set however, so we would expect it to be less reliable due to randomness in the data set.

```
amsv <- sapply(amsv_obj, function(x) x$calc_ams())
if (do_save_outputs) {
  save_fig(partial(plot_amss, amsv_obj, min.max=FALSE),
           path_join(c(fig_dir, "validation-ams-curves.pdf")))
}
# pdf(file=path_join(c(fig_dir, "validation-ams-curves.pdf")))
plot_amss(amsv_obj, min.max=FALSE)
```




```
# dev.off()
```

Summary results

Summary results:

```
get_threshold_ams <- function(ams_mat, G, K, thresholds) {
  ams <- rep(NA, G * K)
  for (g in 1:G) {
    thresh_idx <- which.min(abs(ams_obj[[1]]$thresholds - thresholds[g]))
    model_set <- ((g-1)*K+1):(g*K)
    ams[model_set] <- ams_mat[thresh_idx, model_set]
  }
  return(ams)
}

get_mean_mad_ams <- function(ams, G, K) {
  mads <- rep(NA, G)
  for (g in 1:G) {
    model_set <- ((g-1)*K+1):(g*K)
    mads[g] <- mad(ams[model_set])
  }
  return(mean(mads))
}

result_ams <- get_threshold_ams(ams, G, K, thresholds)
mad_ams <- get_mean_mad_ams(result_ams, G, K)

result_amsv <- get_threshold_ams(amsv, G, 1, thresholds)
```

Table 1: Results for $\lambda = 0.0001$, $G = 3$, $n_{rbf} = 3$, $b = 2$, K=10 on training set ('t') and validation set ('v')

output	AUC	mad.AUC	AMS	mad.AMS
CV OOS	0.869	0.01	3.017	0.168
Test set	0.845		2.738	

```
sprintf("Results for lambda=%.2g, G=%i, n_rbf=%i, K=%i on training set ('%s') and validation set ('%s')"
```

```
## [1] "Results for lambda=0.0001, G=3, n_rbf=3, K=10 on training set ('t') and validation set ('v')"
```

```
sprintf("CV OOS AUC = %.2f ± %.1f", mean(auc), mad(auc))
```

```
## [1] "CV OOS AUC = 0.87 ± 0.0"
```

```
sprintf("CV OOS AMS = %.2f ± %.1f", mean(result_ams), mad_ams)
```

```
## [1] "CV OOS AMS = 3.02 ± 0.2"
```

```
sprintf("Validation set AUC = %.2f", mean(aucv))
```

```
## [1] "Validation set AUC = 0.85"
```

```
sprintf("Validation set AMS = %.2f", mean(result_amsv))
```

```
## [1] "Validation set AMS = 2.74"
```

Save final results to a LaTeX table.

```
library(Hmisc)

# keep top 5 rows
output <- t(matrix(c(mean(auc), mad(auc), mean(result_ams), mad_ams, mean(aucv), NA, mean(result_amsv),
colnames(output) <- c("AUC", "mad.AUC", "AMS", "mad.AMS")
output <- data.frame(output)
output <- mutate_if(output, is.numeric, round, digits=3)
row.names(output) <- c("CV OOS", "Test set")

# Generate LaTeX table
if (do_save_outputs) {
  latex(output, file=path_join(c(dirname(getwd()), "doc/final_results_table.tex")), caption=sprintf("Res
}
```

Analyse Experiments

Import results

```
filepath <- path_join(c(dirname(getwd()), "/output/results_experiments5.csv"))
exp_data <- read.csv(filepath)

# bool to choose whether to save outputs
if (!("do_save_outputs" %in% ls())) {
  do_save_outputs <- FALSE
}
```

Remove some columns that are not important for this analysis and average over any duplicated experiments:

```
exp_data <- exp_data[, !colnames(exp_data) %in% c("c", "Train", "Validation", "model_type")]
exp_data <- exp_data %>% group_by(n_rbf, lambda, poly) %>% summarise_all(funs(mean)) %>% as.data.frame()
```

```
## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
# consider scaling metrics by MAD(metric) to get a handle on variation
scaled_auc <- exp_data[,"auc"]/exp_data[,"mad.auc."]
scaled_ams <- exp_data[,"ams"]/exp_data[,"mad.ams."]
exp_data[, "scaled.auc"] <- scaled_auc
exp_data[, "scaled.ams"] <- scaled_ams
```

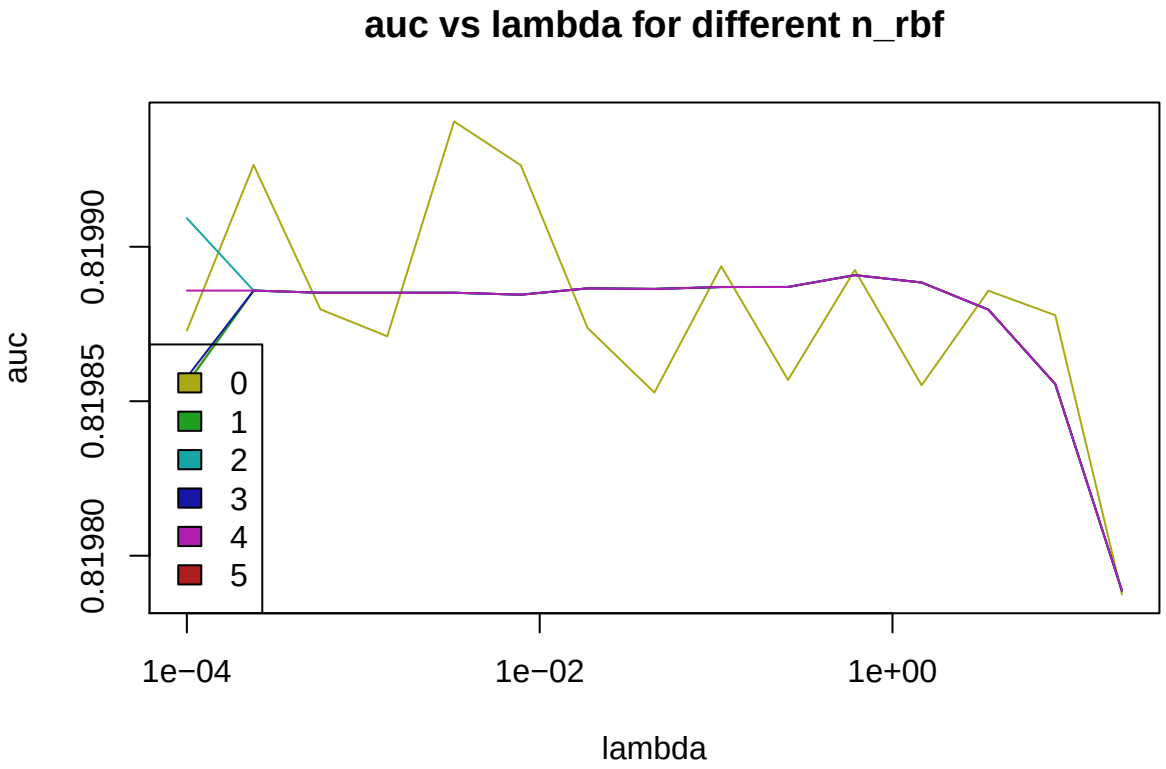
Plot metrics

Implement a convenient plotting function to summarise results.

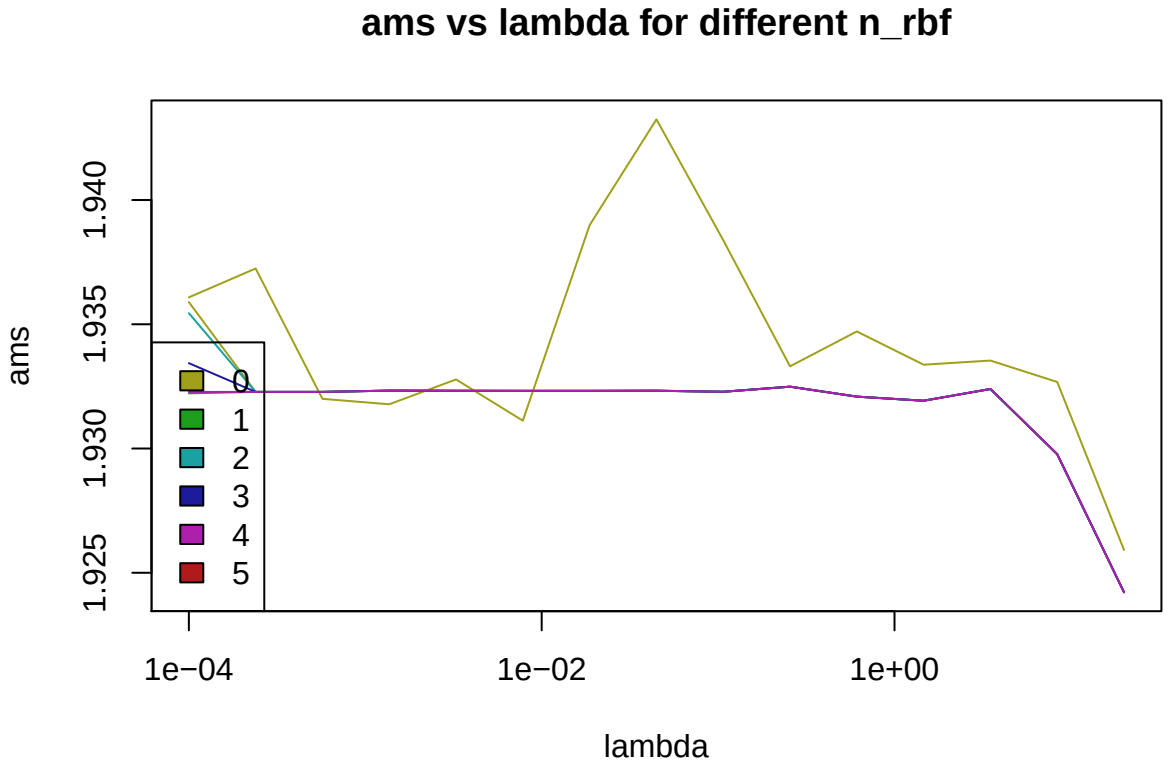
```
# define function to plot against our metric
plot_metric <- function(data, xlabel, ylabel, loop_label=NULL, filt0=TRUE, ...) {
  if (!is.null(loop_label)) {
    init_idx <- min(data[, loop_label])
    ntrials <- max(data[, loop_label])-(-1 + init_idx)
    filt_func <- function(x) data[, loop_label] == x & filt0
  } else{
    ntrials <- 1
    init_idx <- 0
    filt_func <- function(x) filt0
  }

  colours <- generate_colours(ntrials)

  ylim <- c(min(data[filt0, ylabel]), max(data[filt0, ylabel]))

  filt <- filt_func(init_idx)
  plot(data[filt, xlabel], data[filt, ylabel], col=colours[1], type="l", xlab=xlabel, ylab=ylabel, main=
  for (i in 1:ntrials) {
    nr <- i + init_idx
    filt <- filt_func(nr)
    lines(data[filt, xlabel], data[filt, ylabel], col=colours[nr])
  }
  legend("bottomleft", legend=c(init_idx:(ntrials+(init_idx-1))), fill=colours)
}
```

Run plots for various permutations of our model parameters to see their relative performance on our key metrics:

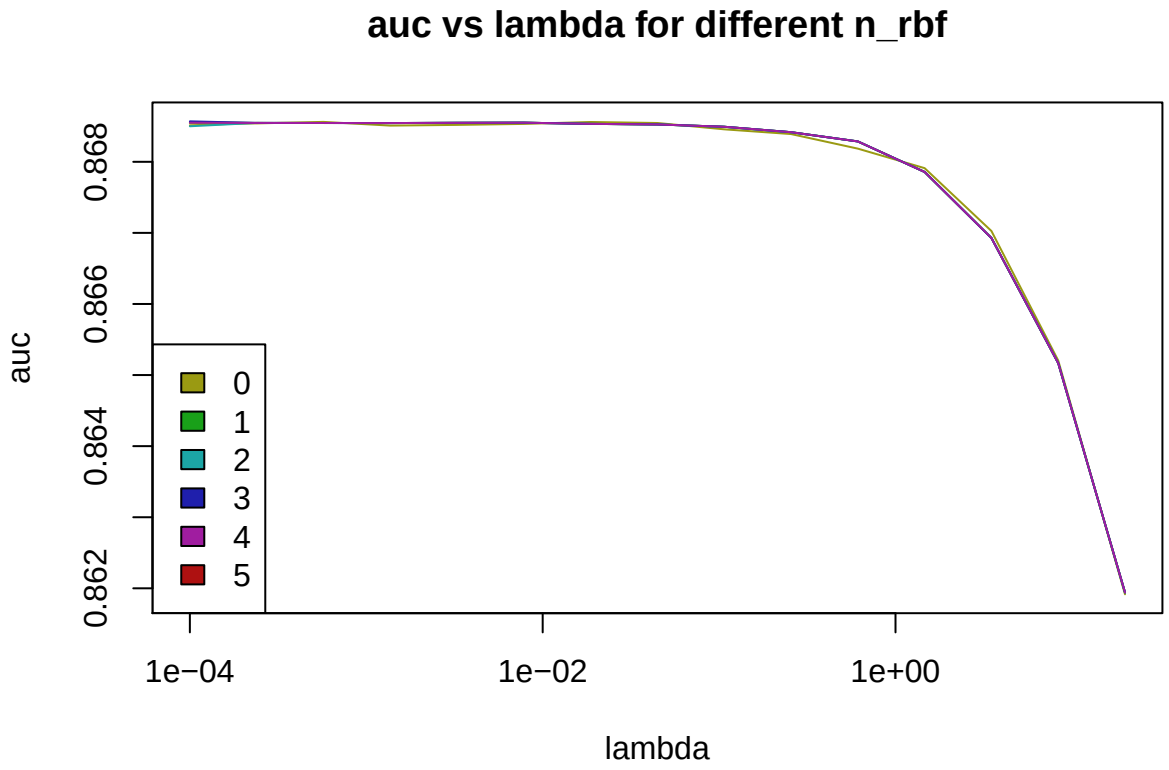
```
# order by our x variable
exp_data <- exp_data[order(exp_data[, "lambda"]), ]

# dir to save figures
fig_dir <- path_join(c(dirname(getwd()), "doc/figs/"))

set_filepath <- function(filename) path_join(c(fig_dir, sprintf("%s.pdf", filename)))

# plot AUC and AMS vs regularisation parameter lambda for different numbers of RBF centroids, for no po
filt0 <- exp_data[, "poly"]==1
if (do_save_outputs) {
  save_fig(partial(plot_metric, exp_data, "lambda", "auc", "n_rbf", filt0, log="x"),
           set_filepath("auc-lambda-by-nrbf-poly=1"))
  save_fig(partial(plot_metric, exp_data, "lambda", "ams", "n_rbf", filt0, log="x"),
           set_filepath("ams-lambda-by-nrbf-poly=1"))
}
plot_metric(exp_data, "lambda", "auc", "n_rbf", filt0, log="x")
```



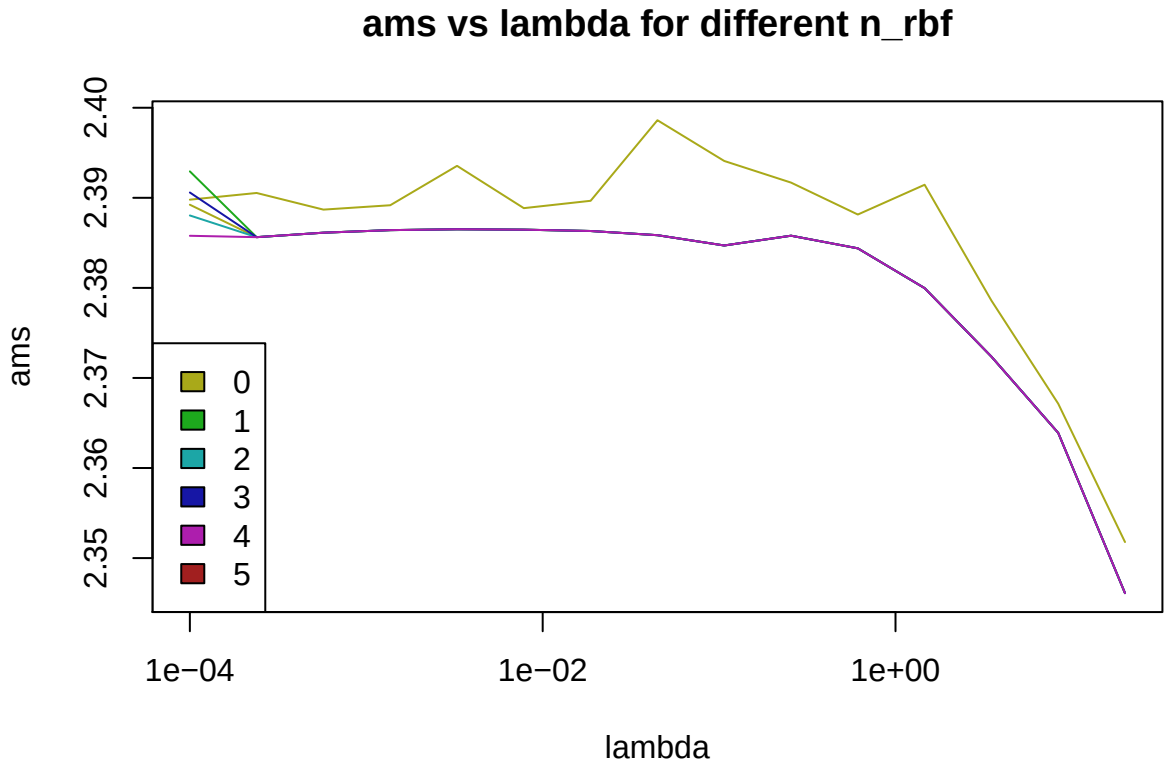

```
plot_metric(exp_data, "lambda", "ams", "n_rbf", filt0, log="x")
```




```
# plot AUC and AMS vs regularisation parameter lambda for different numbers of RBF centroids, for secon
filt0 <- exp_data[, "poly"]==2
if (do_save_outputs) {
  save_fig(partial(plot_metric, exp_data, "lambda", "auc", "n_rbf", filt0, log="x"),
           set_filepath("auc-lambda-by-nrbf-poly=2"))
  save_fig(partial(plot_metric, exp_data, "lambda", "ams", "n_rbf", filt0, log="x"),
           set_filepath("ams-lambda-by-nrbf-poly=2"))
}
plot_metric(exp_data, "lambda", "auc", "n_rbf", filt0, log="x")
```

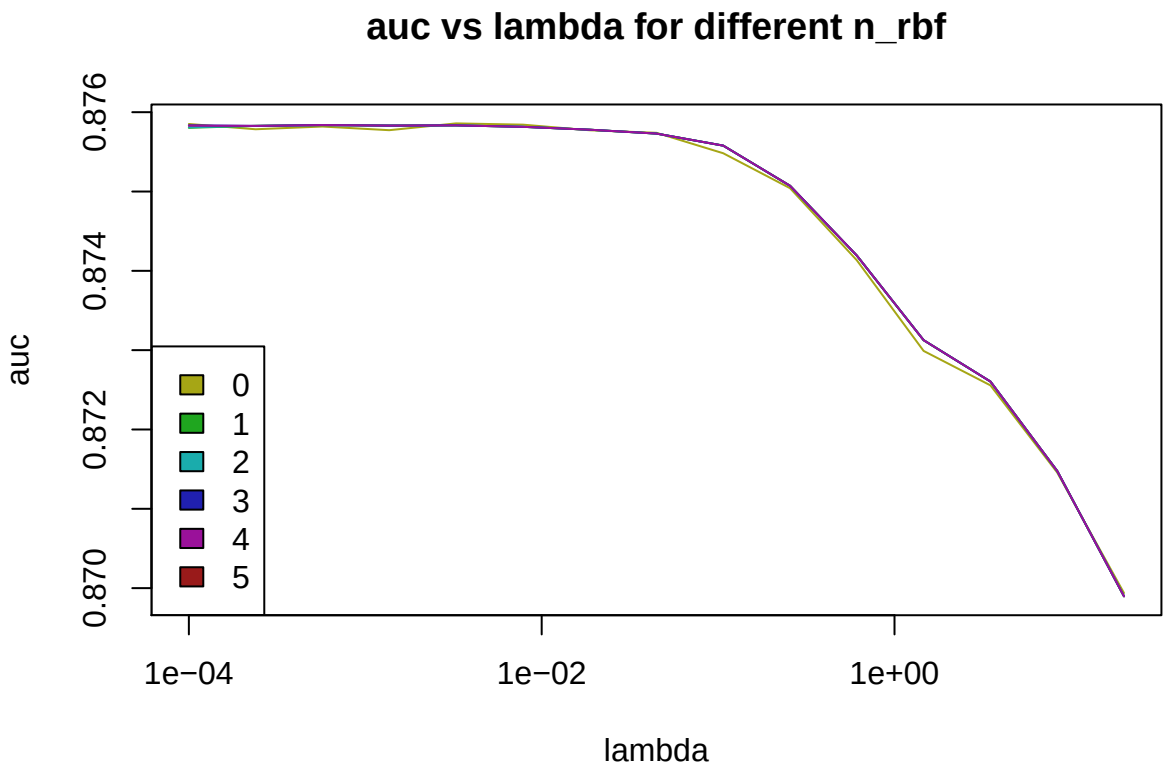



```
plot_metric(exp_data, "lambda", "ams", "n_rbf", filt0, log="x")
```

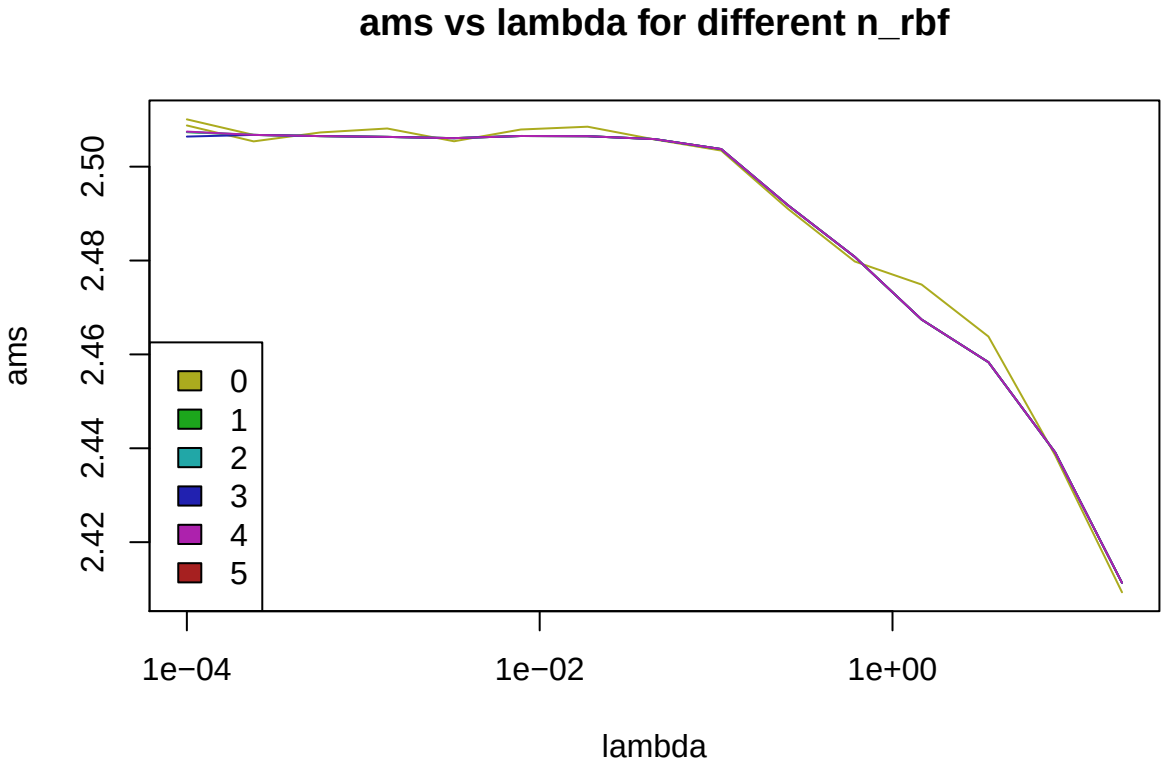



```
# plot AUC and AMS vs regularisation parameter lambda for different numbers of RBF centroids, for 2nd a
filt0 <- exp_data[, "poly"]==3
if (do_save_outputs) {
  save_fig(partial(plot_metric, exp_data, "lambda", "auc", "n_rbf", filt0, log="x"),
```

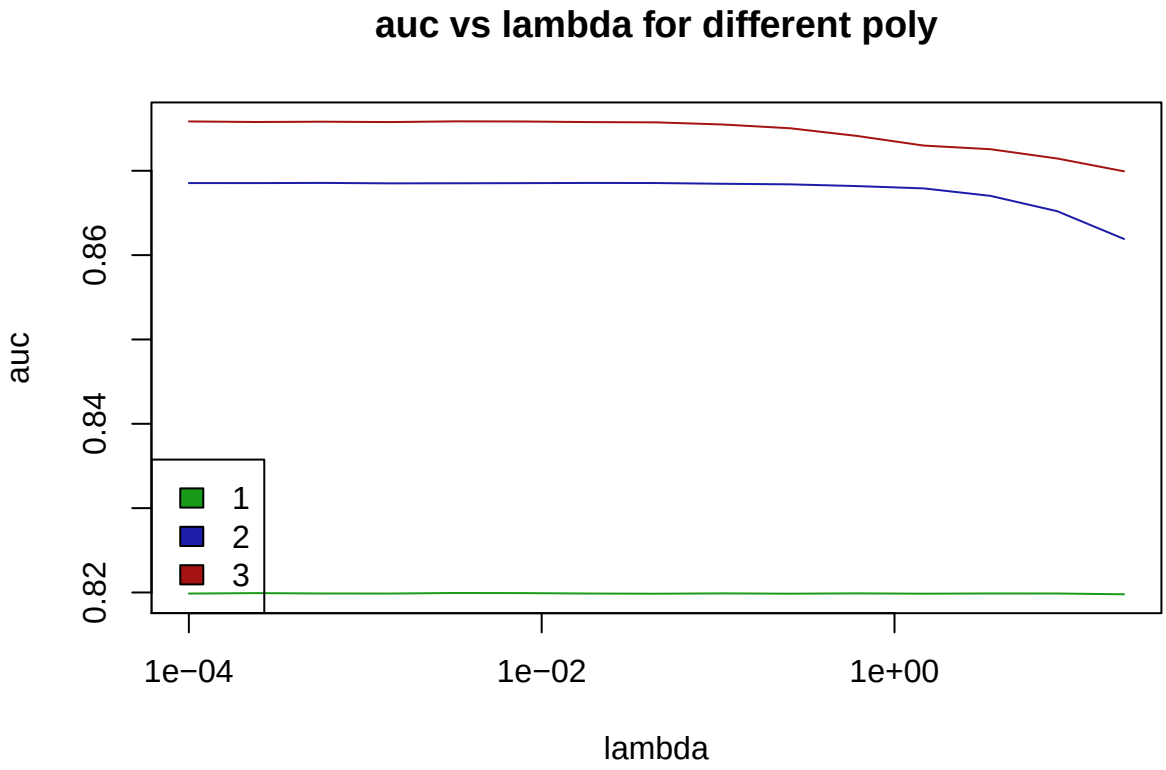
```
            set_filepath("auc-lambda-by-nrbf-poly=3"))
  save_fig(partial(plot_metric, exp_data, "lambda", "ams", "n_rbf", filt0, log="x"),
            set_filepath("ams-lambda-by-nrbf-poly=3"))
}
plot_metric(exp_data, "lambda", "auc", "n_rbf", filt0, log="x")
```



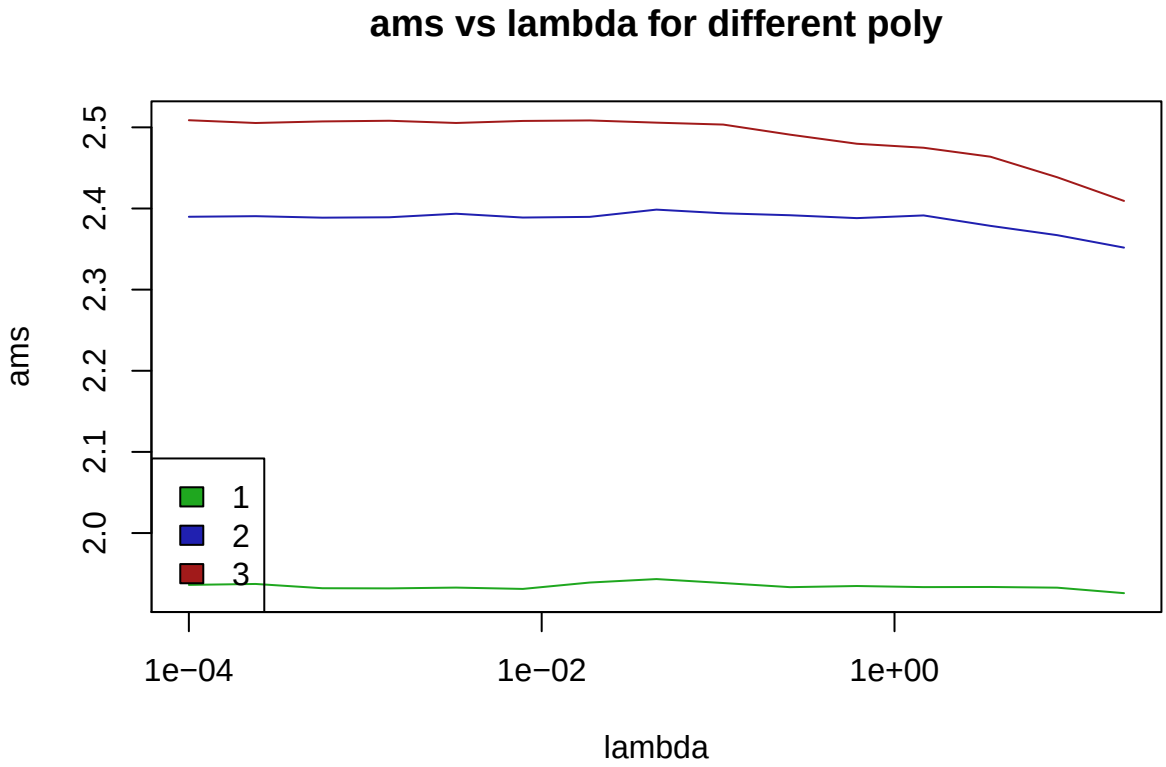

```
plot_metric(exp_data, "lambda", "ams", "n_rbf", filt0, log="x")
```




```
# plot AUC and AMS vs regularisation parameter lambda for different numbers of orders of polynomial tra
filt0 <- exp_data[, "n_rbf"]==0
if (do_save_outputs) {
  save_fig(partial(plot_metric, exp_data, "lambda", "auc", "poly", filt0, log="x"),
           set_filepath("auc-lambda-by-poly-nrbf=0"))
  save_fig(partial(plot_metric, exp_data, "lambda", "ams", "poly", filt0, log="x"),
           set_filepath("ams-lambda-by-poly-nrbf=0"))
}
plot_metric(exp_data, "lambda", "auc", "poly", filt0, log="x")
```

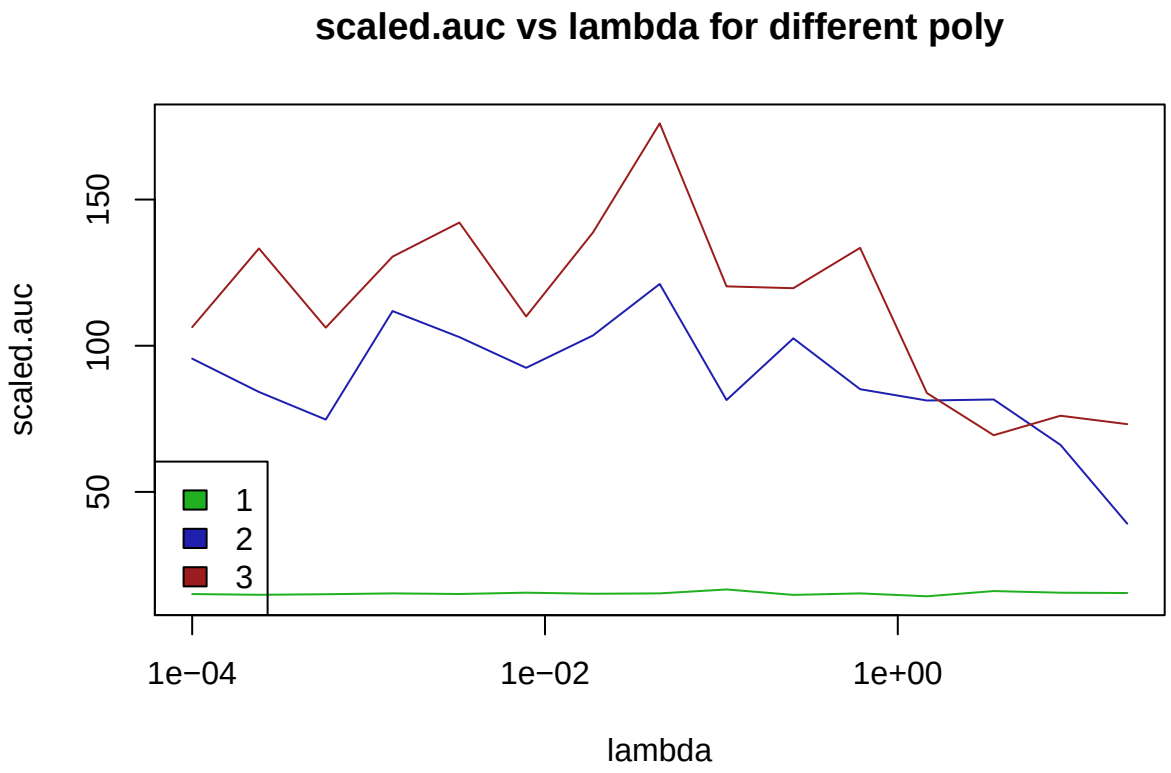



```
plot_metric(exp_data, "lambda", "ams", "poly", filt0, log="x")
```

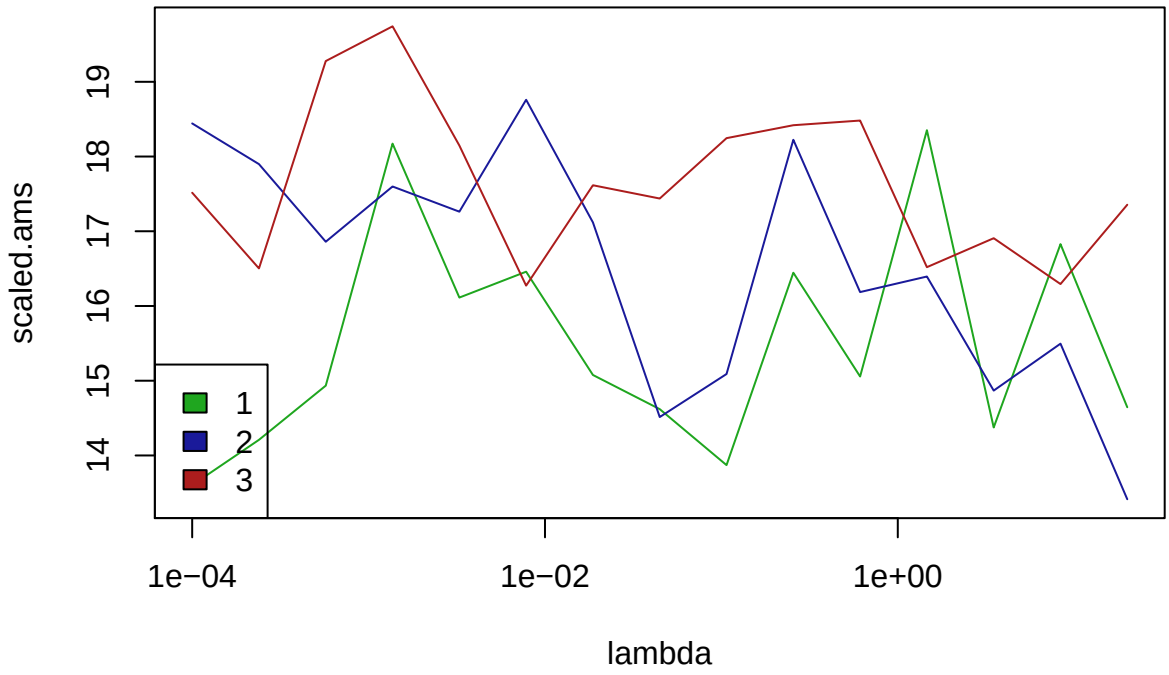



```
# plot AUC and AMS (scaled by respective mean absolute deviation) vs regularisation parameter lambda fo
filt0 <- exp_data[, "n_rbf"]==0
if (do_save_outputs) {
  save_fig(partial(plot_metric, exp_data, "lambda", "scaled.auc", "poly", filt0, log="x"),
```

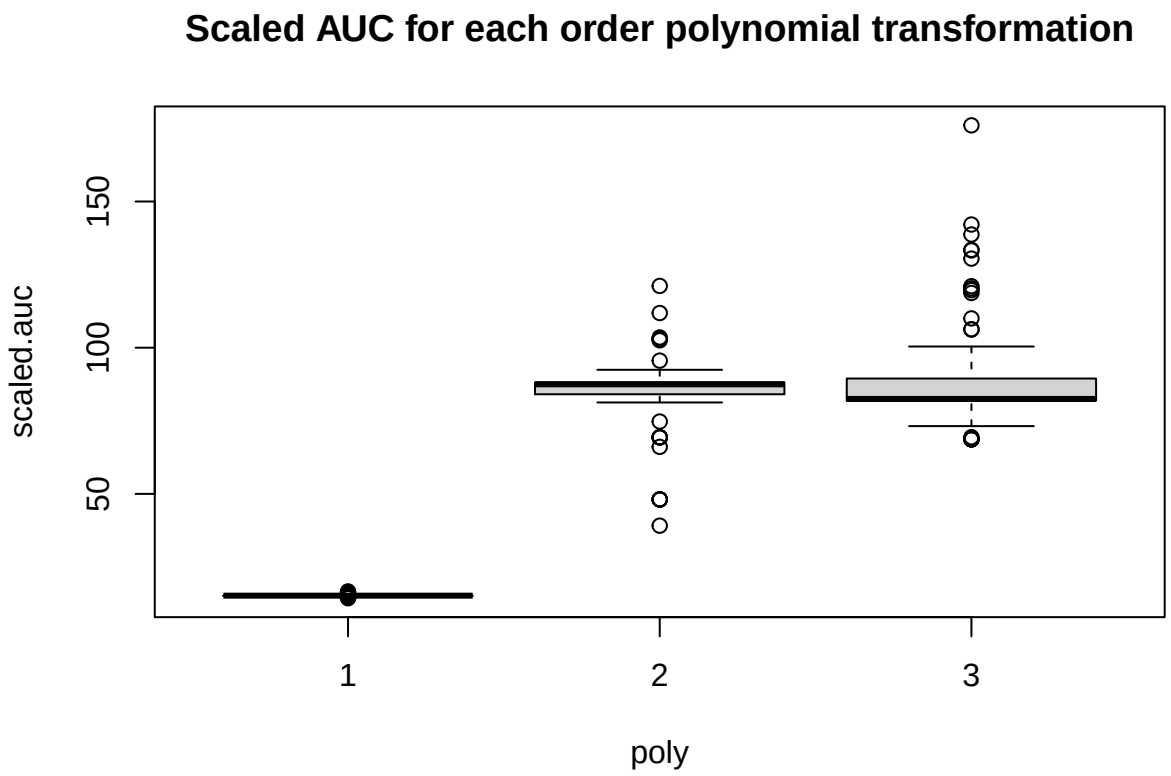
```
          set_filepath("scaledauc-lambda-by-poly-nrbf=0"))
  save_fig(partial(plot_metric, exp_data, "lambda", "scaled.ams", "poly", filt0, log="x"),
          set_filepath("scaledams-lambda-by-poly-nrbf=0"))
}
plot_metric(exp_data, "lambda", "scaled.auc", "poly", filt0, log="x")
```



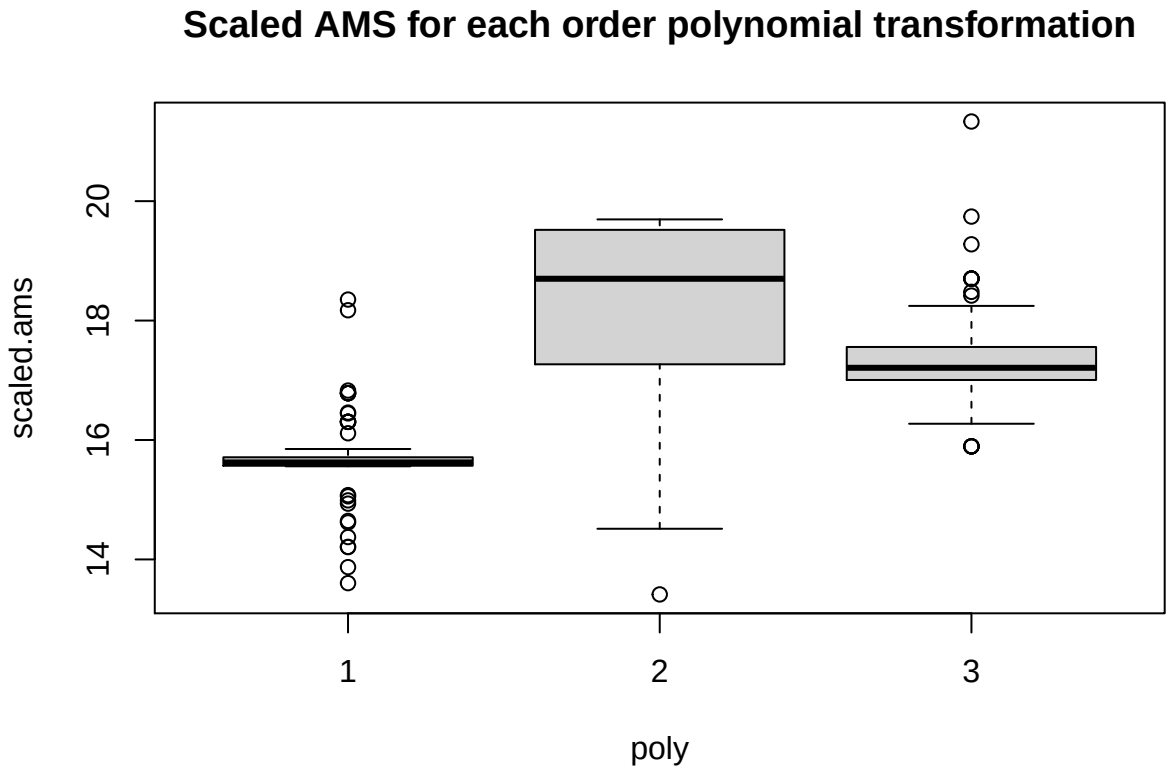

```
plot_metric(exp_data, "lambda", "scaled.ams", "poly", filt0, log="x")
```




```
# plot boxplots of scaled AUC and AMS for each polynomial transformation
if (do_save_outputs) {
  save_fig(partial(boxplot, scaled.auc~poly, data=exp_data, main="Scaled AUC for each order polynomial t
           set_filepath("boxplot-scaledauc-by-poly"))
  save_fig(partial(boxplot, scaled.ams~poly, data=exp_data, main="Scaled AMS for each order polynomial t
           set_filepath("boxplot-scaledams-by-poly"))
}
boxplot(scaled.auc~poly, data=exp_data, main="Scaled AUC for each order polynomial transformation")
```

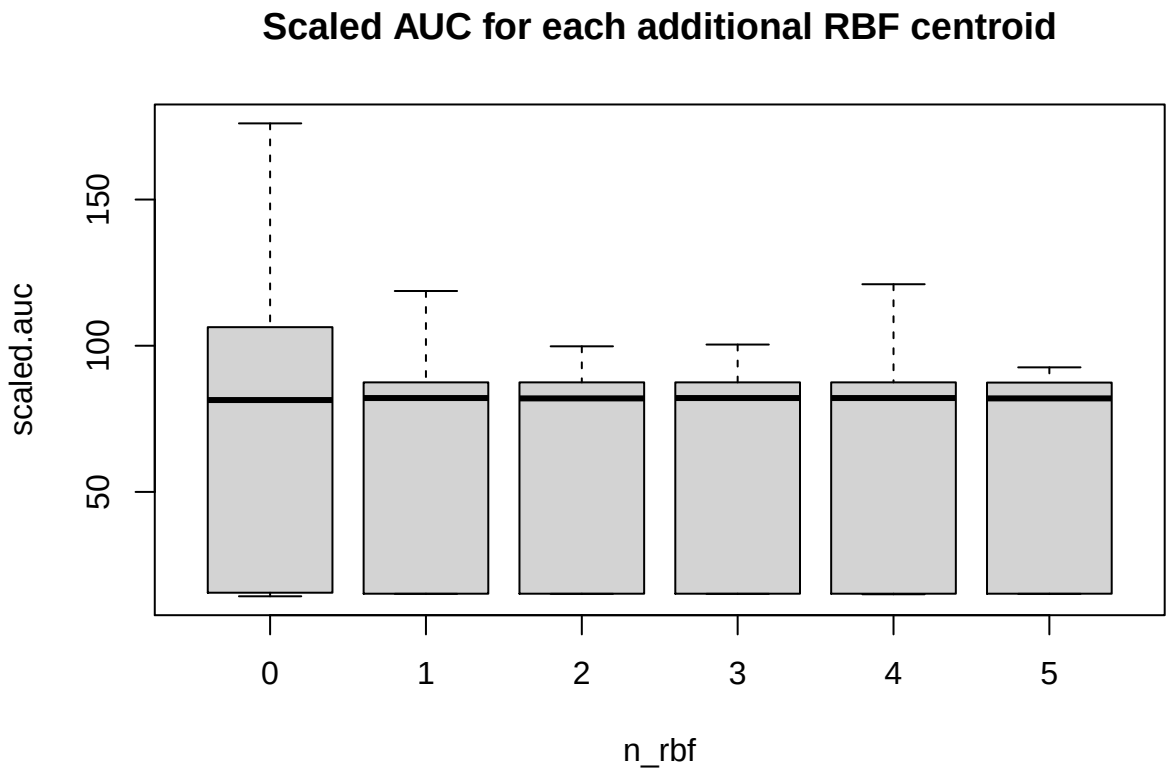



```
boxplot(scaled.ams~poly, data=exp_data, main="Scaled AMS for each order polynomial transformation")
```

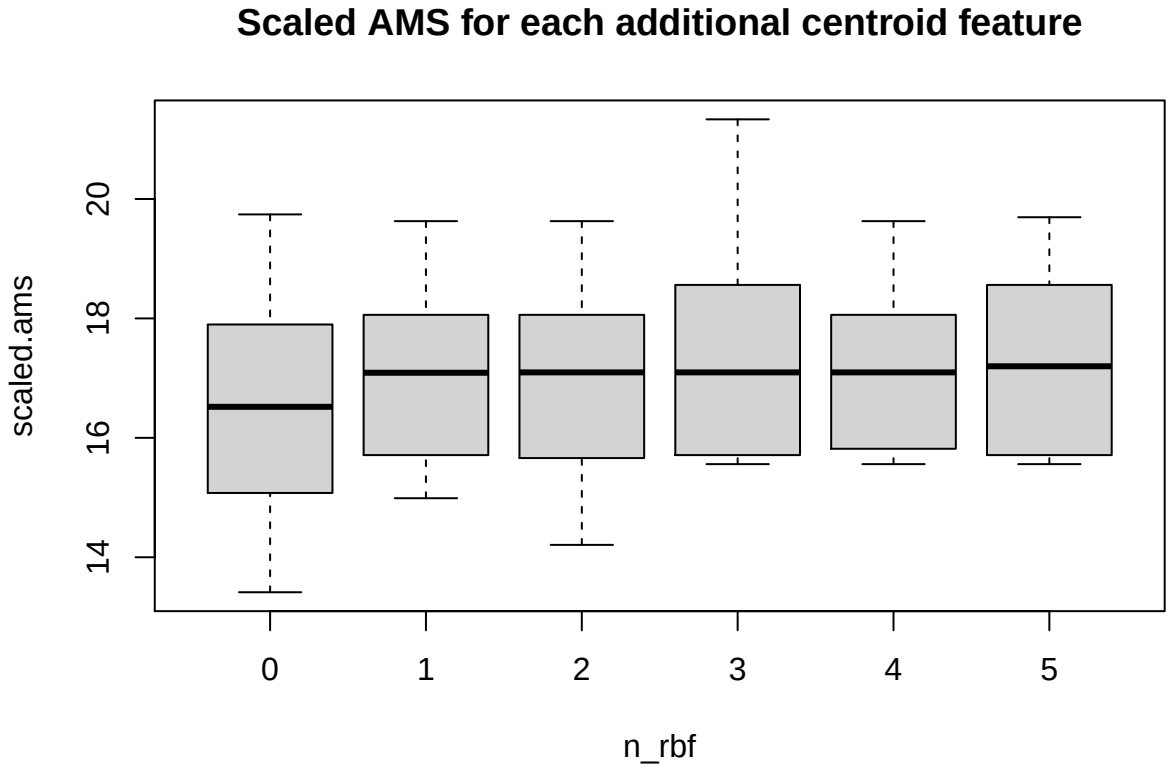



```
# plot boxplots of scaled AUC and AMS for additional RBF centroid
if (do_save_outputs) {
  save_fig(partial(boxplot, scaled.auc~n_rbf, data=exp_data, main="Scaled AUC for each additional RBF c
```

```
            set_filepath("boxplot-scaledauc-by-nrbf"))
  save_fig(partial(boxplot, scaled.ams~n_rbf, data=exp_data, main="Scaled AMS for each additional RBF c
            set_filepath("boxplot-scaledams-by-nrbf"))
}
boxplot(scaled.auc~n_rbf, data=exp_data, main="Scaled AUC for each additional RBF centroid")
```




```
boxplot(scaled.ams~n_rbf, data=exp_data, main="Scaled AMS for each additional centroid feature")
```



##

Record results

Print results as a LaTeX table that can be imported into a report easily. Not too surprisingly we see different models appear at the top of our list depending on how we decide to choose the best model. Since our goal is ultimately to maximise AMS, it makes sense that this (rather than AUC) should be the metric we choose, despite the higher variance it displays. To try and mitigate this variance, we can choose to sort by a scaled version of AMS, where we take the mean absolute deviation across the folds and scale by that, to try and optimise for the best result with the least variation and therefore hopefully the best generalisation.

Surprisingly, the top model appears to have the following set of parameters: $n_{rbf} = 2$, $\lambda = 1e - 4$ and the inclusion of upto 3rd order polynomial terms.

```
library(Hmisc)
#scaled <- head(arrange(exp_data, scaled_ams, scaled_auc))

# sort/filter data to only retain what we want to record and round numerical values to 3d.p for readibi
output <- exp_data[, c("n_rbf", "lambda", "poly", "auc", "mad.auc.", "ams", "mad.ams.", "scaled.auc", "
  arrange(desc(scaled.ams), desc(scaled.auc)) %>%
  mutate_if(is.numeric, round, digits=3)

# keep top 5 rows
output <- output[1:5, ]
print(output)
```

```
##   n_rbf lambda poly   auc mad.auc.   ams mad.ams. scaled.auc scaled.ams
## 1     3  0.000    3 0.876    0.009 2.507    0.118    100.399     21.334
## 2     0  0.001    3 0.876    0.007 2.508    0.127    130.466     19.742
## 3     5  0.000    2 0.869    0.010 2.386    0.121     87.225     19.695
## 4     1  0.000    2 0.869    0.010 2.386    0.122     87.087     19.629
## 5     2  0.000    2 0.869    0.010 2.386    0.122     87.087     19.629
```

Table 2: Results table of the top 5 experiments

output	n_{rbf}	lambda	poly	auc	mad_{auc}	ams	mad_{ams}	$scaled_{auc}$	$scaled_{ams}$
1	3	0.000	3	0.876	0.009	2.507	0.118	100.399	21.334
2	0	0.001	3	0.876	0.007	2.508	0.127	130.466	19.742
3	5	0.000	2	0.869	0.010	2.386	0.121	87.225	19.695
4	1	0.000	2	0.869	0.010	2.386	0.122	87.087	19.629
5	2	0.000	2	0.869	0.010	2.386	0.122	87.087	19.629

```
# make column names latex friendly
colnames(output) <- sub("(\\w+)\\.(\\w+)\\.?", "\\$\\1_\\{\\2\\}\\$", colnames(output))
colnames(output) <- sub("n\\_rbf", "\\$n_{rbf}\\$", colnames(output))

# Generate LaTeX table
if (do_save_outputs) {
  latex(output, file=path_join(c(dirname(getwd()), "doc/results_table.tex")), caption="Results table of
}
```