2. SVM

SVM

In the `lhc` package we have implemented two support vector machine (SVM) algorithms: `svm` and `kernel_svm`. In this section of the notebook we attempt to apply these methods to the Higgs boson classification problem.

It is expected that soft-margin SVM will have a similar performance to logistic regression. Kernel SVM could improve performance by transforming the data and creating a non linear decision boundary. Unfortunately, SVM algorithms do not scale well with n and so we are unlikely to be able to train our models on a significant portion of the training data.

Our SVM functions take as inputs: an $n \times d$ design matrix $\mathbf{X}$, binary class labels $\mathbf{y} \in \{-1, 1\}^n$, a parameter C to controls the relative weighting of maximising the margin vs minimising the slack variables. For `kernel_svm` a kernel function k is also needed.

The functions use the function `solve.QP` from the `quadprog` package to solve the Lagrangian optimisation problem:

$$L(\boldsymbol{\lambda}) = -\frac{\boldsymbol{\lambda}((\mathbf{y}\mathbf{y}^T) \circ (\mathbf{X}\mathbf{X}^T))\boldsymbol{\lambda}}{2} + \langle \mathbf{1}, \boldsymbol{\lambda} \rangle$$

subject to

$$0 \leq \lambda_i \leq C, \qquad \sum_i \lambda_i y_i = 0$$

For `kernel_svm`, $\mathbf{X}\mathbf{X}^T$ is replaced with $\mathbf{K}$, where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

Finally, the functions return a function/trained model which predicts the classes of new samples. For example, the prediction function for the soft-margin SVM takes the form $y_i = \mathbf{w}^T\mathbf{x}_i + w_0$.

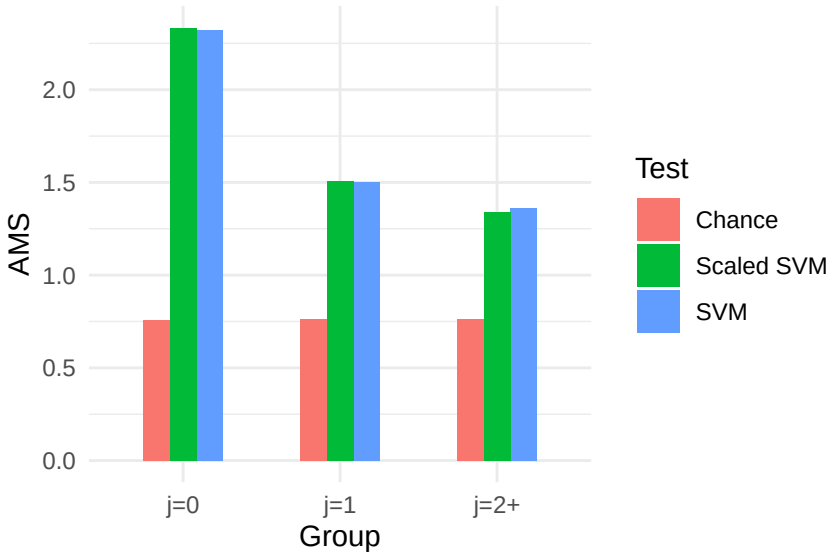
SVM algorithms return the predicted class of samples $\hat{y}_i$ not the probability $P(y_i = s|\mathbf{x}_i)$. This means we cannot us our `ROC_curve` and `AMS_data` object classes which vary a decision threshold, and we so calculate a single AMS value for the test set.

Soft-margin SVM

First we run a first pass of the soft-margin SVM on each of our groups using a small training set. We test how standardising the data before fitting affects performance, and how performance compares to chance (random assignment of classes).

```
#for SVM recode Y as -1 and 1
info$Y[info$Y == 0] <- -1

#Set seed so random samples are reproducible
set.seed(110)

#create a results table to store AMS
results <- as.data.frame(matrix(NA, nrow=G, ncol=4))
results[,1] <- groups
colnames(results) <- c("Group", "SVM", "Scaled SVM", "Chance")

#loop through the groups
n <- 500
for(g in 1:G){
  #select samples from this group
```

```r
  X_group <- X[info$Group==groups[[g]], !colnames(X) %in% features_to_rm[[g]]]
  info_group <- info[info$Group==groups[[g]], ]

  #randomly select n sample to train, and use the rest as test
  ind <- c(rep(0, n), rep(1, nrow(X_group) - n))
  kI <- sample(ind, nrow(X_group))

  X_train <- X_group[kI != 1,]
  y_train <- info_group[kI != 1, "Y"]

  X_test <- X_group[kI == 1,]
  y_test <- info_group[kI == 1, "Y"]
  w_test <- info_group[kI == 1, "Weight"]

  #train a model on unscaled data
  model <- svm(X_train, y_train, C=1)

  #use the resulting function to predict the classes of the training samples
  y_pred <- model(X_test)

  #calculate AMS
  results[g, 2] <- ams_metric(y_test, y_pred, w_test)

  #scale the training data, and scale the test data with the same transformation
  X_test_scaled <- scale_dat(X_test, X_train, add.intercept = F)
  X_train_scaled <- scale_dat(X_train, X_train, add.intercept = F)

  #train a model on scaled data and calculate AMS
  model <- svm(X_train_scaled, y_train, C=1)
  y_pred_scaled <- model(X_test_scaled)
  results[g, 3] <- ams_metric(y_test, y_pred_scaled, w_test)

  #randomly assign classes and calculate AMS
  y_rand <- sample(c(-1, 1), length(y_test), replace=T)
  results[g, 4] <- ams_metric(y_test, y_rand, w_test)
}

results %>%
  pivot_longer(cols=-"Group", names_to="Test", values_to="AMS") %>%
  ggplot(aes(x=Group, y=AMS, fill=Test)) +
    geom_bar(position="dodge", stat = "identity", width=0.5) +
    theme_minimal()
```



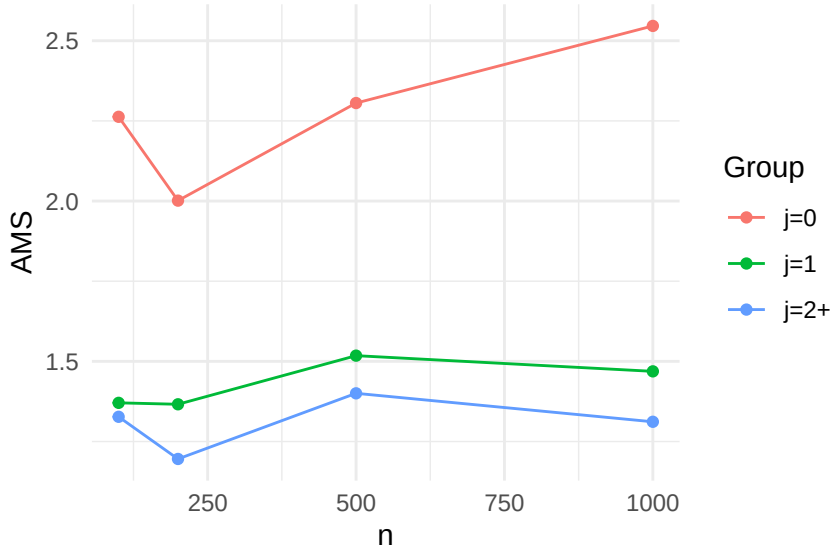
Interestingly, the SVM models trained on only 500 samples perform almost as well as the logistic regression. It's likely that this is around the best performance possible with a linear boundary. Scaling the data before training appears to have a negligible impact.

Training size

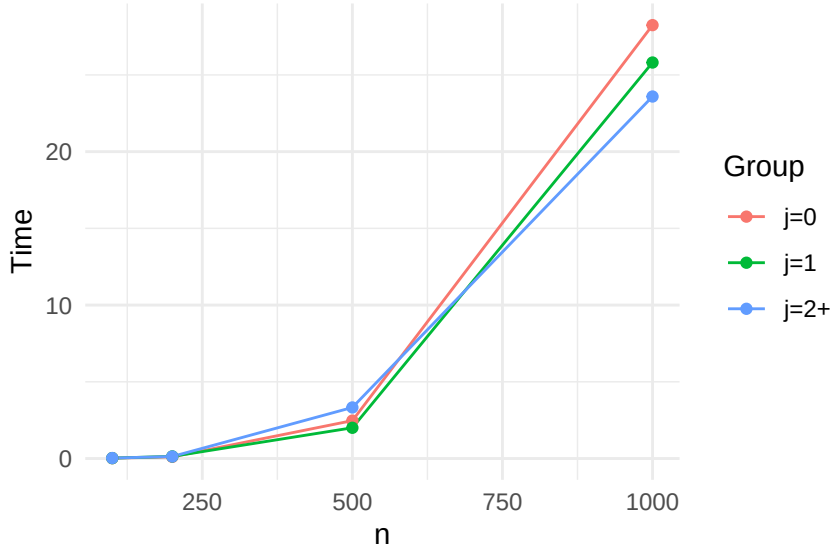
How does increasing n affect time and performance?

```
#create a results table to store AMS
sizes <- c(100, 200, 500, 1000)
results <- as.data.frame(matrix(NA, nrow=G, ncol=length(sizes)+1))
results[,1] <- groups
colnames(results) <- c("Group", sizes)

#and a table to store the times
times <- results

for(i in 1:length(sizes)){
  n <- sizes[i]
  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]], ]

    ind <- c(rep(0, n), rep(1, nrow(X_group)-n))
    kI <- sample(ind, nrow(X_group))

    X_train <- X_group[kI == 0,]
    y_train <- info_group[kI == 0, "Y"]

    X_test <- X_group[kI == 1,]
    y_test <- info_group[kI == 1, "Y"]
    w_test <- info_group[kI == 1, "Weight"]

    time_check<- function(){
      model <- svm(X_train, y_train, C=1)
      y_pred <- model(X_test)
      return(y_pred)
    }
```

```
    times[g, i+1] <- system.time(y_pred <- time_check())["elapsed"]

    results[g, i+1] <- ams_metric(y_test, y_pred, w_test)
  }
}

results %>%
  pivot_longer(cols=-"Group", names_to="n", values_to="AMS") %>%
  mutate(n=as.numeric(n)) %>%
  ggplot(aes(x=n, y=AMS, colour=Group)) +
    geom_line() +
    geom_point() +
    theme_minimal()
```




```
times %>%
  pivot_longer(cols=-"Group", names_to="n", values_to="Time") %>%
  mutate(n=as.numeric(n)) %>%
  ggplot(aes(x=n, y=Time, colour=Group)) +
    geom_line() +
    geom_point() +
    theme_minimal()
```



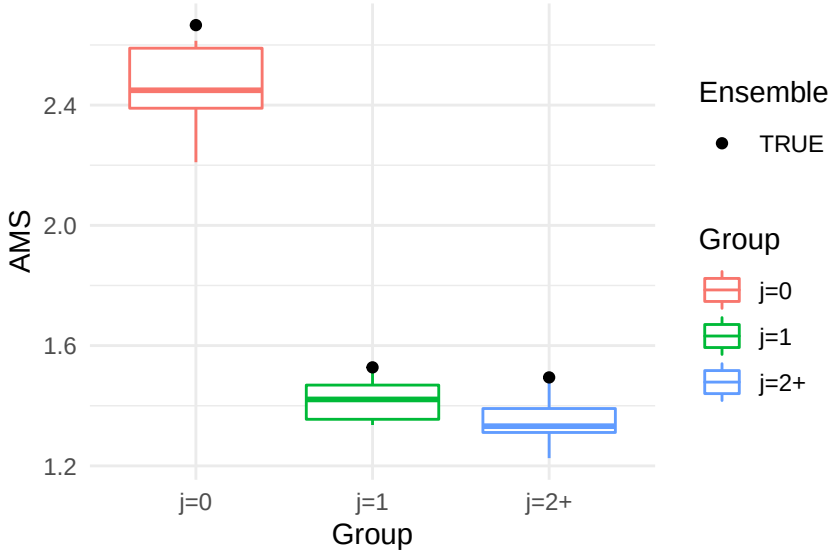
We can see that performance (AMS score) does not change much when the sample size is doubled from 500 to 1000, but the training time increases with approximately $\mathcal{O}(n^3)$. The group with no jets seems to perform consistently better than the other two groups.

Ensemble SVM

Ensemble SVM is a technique where multiple SVM models are trained on subsets of the training data. To classify new samples, the majority vote from the models is taken. Dividing a dataset of n samples over m models reduces the computational complexity of an $\mathcal{O}(n^2)$ algorithm to $\mathcal{O}(n^2/m)$.

```
m <- 10
n <- 500

results <- as.data.frame(matrix(NA, nrow=G, ncol=m+2))
results[,1] <- groups
colnames(results) <- c("Group", paste("Model", 1:m), "Ensemble")

#use a different parameter C for each model
params <- sample(1:50, m)

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]], ]

  ind <- c(rep(1:m, n), rep(0, nrow(X_group)-m*n))
  kI <- sample(ind, nrow(X_group))

  X_test <- X_group[kI==0, ]
  y_test <- info_group[kI==0, "Y"]
  w_test <- info_group[kI==0, "Weight"]

  #store test predictions from m models
  y_predictions <- matrix(NA, ncol=m, nrow=nrow(X_test))

  #for each of the k groups, fit a svm, and calc ams
  for(i in 1:m){
    X_train <- X_group[kI == i,]
    y_train <- info_group[kI == i, "Y"]

    model <- svm(X_train, y_train, C=params[i])
    y_pred <- model(X_test)
    y_predictions[,i] <- y_pred

    results[g, i+1] <- ams_metric(y_test, y_pred, w_test)
  }

  #now find the ams from majority vote
  y_pred <- sign(rowMeans(y_predictions, na.rm=T))
  results[g, m+2] <- ams_metric(y_test, y_pred, w_test)
}


plot_data <- results %>%
  pivot_longer(cols=-"Group", names_to="Model", values_to="AMS") %>%
  mutate(Ensemble = grepl("Ensemble", Model))

ggplot(plot_data[!plot_data$Ensemble,], aes(x=Group, y=AMS, colour=Group)) +
```

```
  geom_boxplot() +
  geom_point(data=plot_data[plot_data$Ensemble,], aes(fill=Ensemble), colour="black") +
  theme_minimal()
```



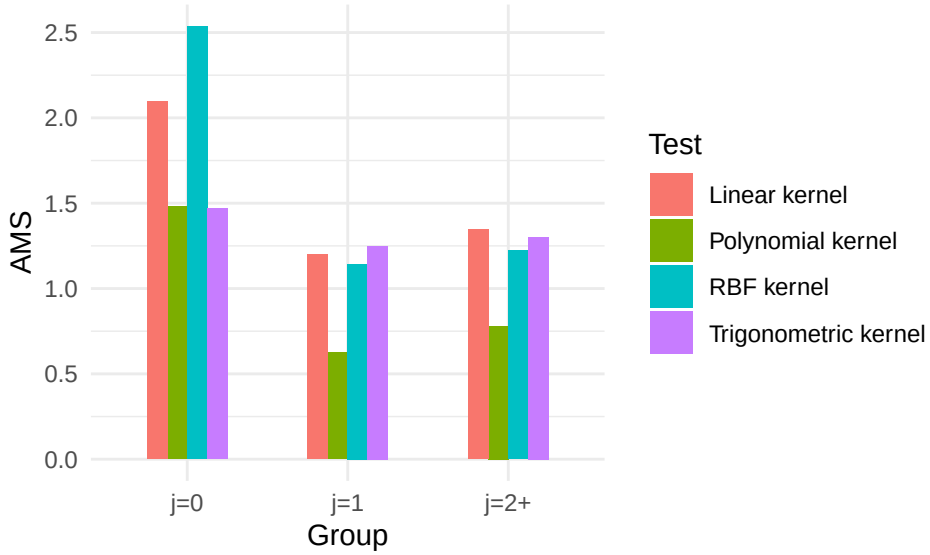
The ensemble SVM outperforms the individual models and and scales better with n, however the improvements in AMS are fairly modest.

Kernel SVM

Running a small kernel SVM on each group. Unfortunately our implementation scales badly with n and so only small training and test sets can be used (predictions are also computational expensive).

```
train_n <- 200
test_n <- 2000

kernels <- list(
  "Polynomial kernel" = tuned_kernel(poly_kernel, b=2),
  "RBF kernel" = tuned_kernel(rbf_kernel, sigma=10),
  "Trigonometric kernel" = tuned_kernel(trig_kernel, b=5),
  "Linear kernel" = tuned_kernel(lin_kernel)
)

results <- as.data.frame(matrix(NA, nrow=G, ncol=length(kernels)+1))
results[,1] <- groups
colnames(results) <- c("Group", names(kernels))

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]], ]

  for(k in 1:length(kernels)){
    #create index training samples = 1, test samples = 2, other = 0
    ind <- c(rep(1, train_n), rep(2, test_n), rep(0, nrow(X_group)-train_n-test_n))
    kI <- sample(ind, nrow(X_group))

    X_train <- X_group[kI==1, ]
    y_train <- info_group[kI==1, "Y"]

    X_test <- X_group[kI==2, ]
```

```
    y_test <- info_group[kI==2, "Y"]
    w_test <- info_group[kI==2, "Weight"]

    #scale data so values in poly kernel aren't too large
    X_test_scaled <- scale_dat(X_test, X_train, add.intercept = F)
    X_train_scaled <- scale_dat(X_train, X_train, add.intercept = F)

    ckernel <- kernels[[k]]
    model <- kernel_svm(X_train_scaled, y_train, C=1, ckernel)
    y_pred <- model(X_test_scaled)
    results[g, k+1] <- ams_metric(y_test, y_pred, w_test)
  }
}

results %>%
  pivot_longer(cols=-"Group", names_to="Test", values_to="AMS") %>%
  ggplot(aes(x=Group, y=AMS, fill=Test)) +
    geom_bar(position="dodge", stat = "identity", width=0.5) +
    theme_minimal()
```



The RBF, trigonometric and polynomial kernels perform quite similarly in this trial. To improve performance we could do a grid search for the optimal hyperparameters with cross validation, however we are restricted by the time complexity of the kernel SVM function and so we instead focus on feature engineering the logistic regression model.