Factor Target Gene Prediction through Multi-Omics Datasets

A. Kapoor, D. Langreiter, U. Samant

Introduction

Aim:
The aim of this project is to apply classification techniques to predict novel transcription factor target genes, focusing on Sox2 and Nanog during embryonic stem cell (ESC) differentiation.

Background:
Transcriptional regulation is a fundamental ess in all living organisms, driven by transcription factors that control mRNA expression. These transcriptional networks are crucial in development, lineage specification, and cell fate decisions during early embryonic development (Theunissen and Jaenisch, 2017). Recent advances in omics technologies allow for the profiling of genome-wide transcriptional and epigenetic events, providing a deeper understanding of these networks.

In this project, we utilize high-temporal-resolution multi-omics data of ESC differentiation (Yang et al., 2019) to predict novel substrates of Sox2 and Nanog—two key transcription factors involved in maintaining pluripotency and guiding cell differentiation.

Dataset Overview: - **Transcriptome:** Time-course mRNA profiles during ESC differentiation. - **Proteome:** Time-course protein expression profiles during ESC differentiation. - **Epigenome:** Time-course ESC differentation epigonme profiles of 6 histone marks.

We will develop and validate a classification model to predict novel transcription factor target genes, focusing on Sox2 and Nanog, using the provided multi-omics datasets.

Load Required Libraries and Data

We start by loading the necessary R packages and the dataset `Final_Project_ESC.RData`, which contains the transcriptome, proteome, and epigenome data, along with a subset of known Sox2/Nanog target genes.

```r
# Load necessary packages and data
load("Final_Project_ESC.RData", verbose = TRUE)
```

```
## Loading objects:
##   Transcriptome
##   Proteome
##   H3K4me3
##   H3K27me3
##   PolII
##   H3K4me1
##   H3K27ac
##   H3K9me2
##   cMyc_target_genes
##   cMyc_target_genes_subset
##   OSN_target_genes
##   OSN_target_genes_subset
```

```r
suppressPackageStartupMessages({
    library(e1071)
    library(ggplot2)
    library(ROCR)
    library(calibrate)
    library(dplyr)
    library(tibble)
    library(reshape2)
    library(kernlab)
    library(caret)
    library(randomForest)
    library(adabag)
    library(gbm)
    library(xgboost)
    library(nnet)
    library(pROC)
    library(doParallel)
    library(calibrate)
})

set.seed(123)
```

Describe and explore the Data set details

Before beginning data analysis it is important understand and investigate the data. The goal of this report is to predict to predict novel transcription factor target genes from multi-omics data. For each of our datasets, one can look at the structure of the data and perform PCA Analysis as means of identifying trends in the dataset.

Transcriptome

```r
head(Transcriptome)
```

```
##                 0hr         1hr          6hr        12hr        24hr
## GNAI3  8.881784e-16  0.29131114  0.618947976  0.48337689  0.71864514
## CDC45  0.000000e+00  0.25062323  0.199000752  0.38800303  0.47163966
## H19    0.000000e+00  0.38475910  0.471216601  0.55341685  1.87279375
## SCML2  8.881784e-16  0.64099671  0.917100751  0.73622935  0.74708761
## NARF  -8.881784e-16 -0.40736349 -1.186942510 -0.69986049 -0.15754727
## CAV2   5.551115e-17  0.01608049  0.002154149  0.07278802  0.06737809
##              36hr        48hr        72hr
## GNAI3  0.90833508  0.63408090  0.81560693
## CDC45  0.20338260 -0.03095025 -0.55159826
## H19    2.77535428  4.17558703  4.34079023
## SCML2  0.71202112 -0.20856228 -1.12588475
## NARF  -0.73012533 -0.11611985 -0.01534405
## CAV2   0.08131677  0.46840687  1.32719756
```

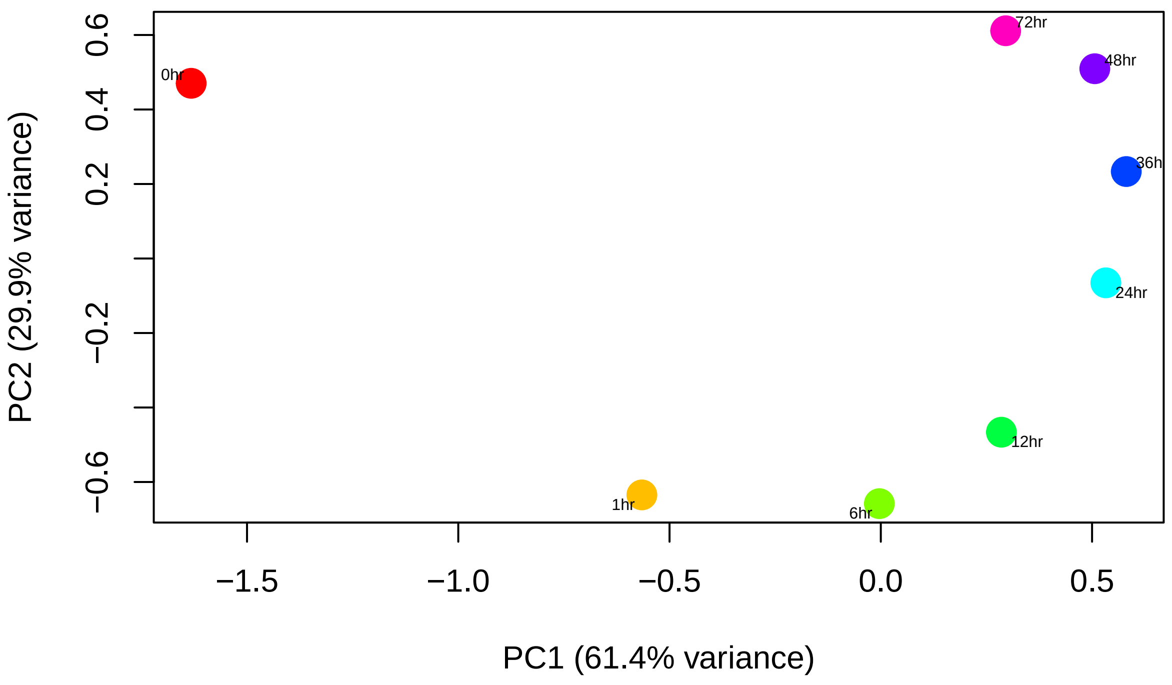
```r
dim(Transcriptome)
```

```
## [1] 19788     8
```

```r
colnames(Transcriptome)
```

```
## [1] "0hr"  "1hr"  "6hr"  "12hr" "24hr" "36hr" "48hr" "72hr"
```

```
# PCA analysis on the correlation matrix of the transcriptome data
cor.mat <- cor(Transcriptome)
pca.mat <- prcomp(cor.mat)

# Plot the PCA
grp <- rownames(pca.mat$x)
grp.col <- rainbow(nrow(pca.mat$x))
names(grp.col) <- rownames(pca.mat$x)

# Generate PCA plot
plot(pca.mat$x[,1], pca.mat$x[,2], col=grp.col[grp], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.mat)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.mat)$importance[2,2]*100,1), "% variance)"))

# Add sample labels to the plot
calibrate::textxy(pca.mat$x[,1], pca.mat$x[,2], labs=grp, cex=0.5)
```

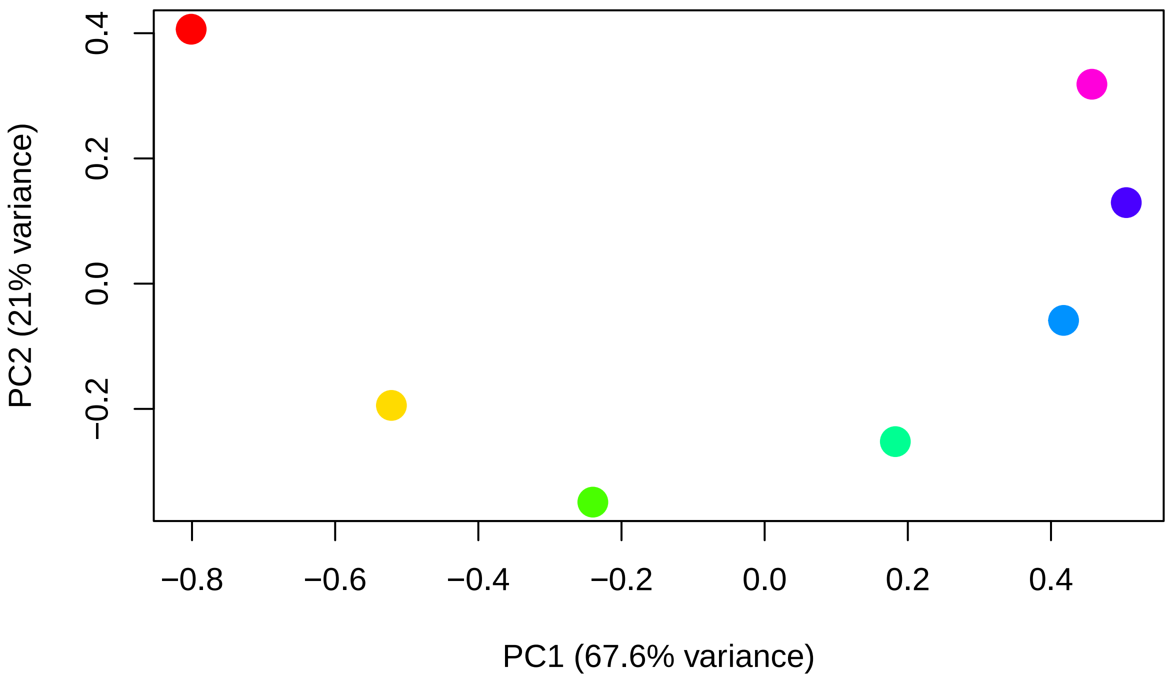


Proteome

```
cor.proteome <- cor(Proteome)
pca.proteome <- prcomp(cor.proteome)
summary(pca.proteome)$importance
```

```
##                              PC1       PC2       PC3       PC4        PC5
## Standard deviation     0.5233315 0.2915203 0.1624012 0.1053089 0.07579296
## Proportion of Variance 0.6763300 0.2098700 0.0651300 0.0273900 0.01419000
## Cumulative Proportion  0.6763300 0.8862000 0.9513300 0.9787100 0.99290000
##                               PC6          PC7
## Standard deviation     0.05362587 4.481316e-17
## Proportion of Variance 0.00710000 0.000000e+00
## Cumulative Proportion  1.00000000 1.000000e+00
```

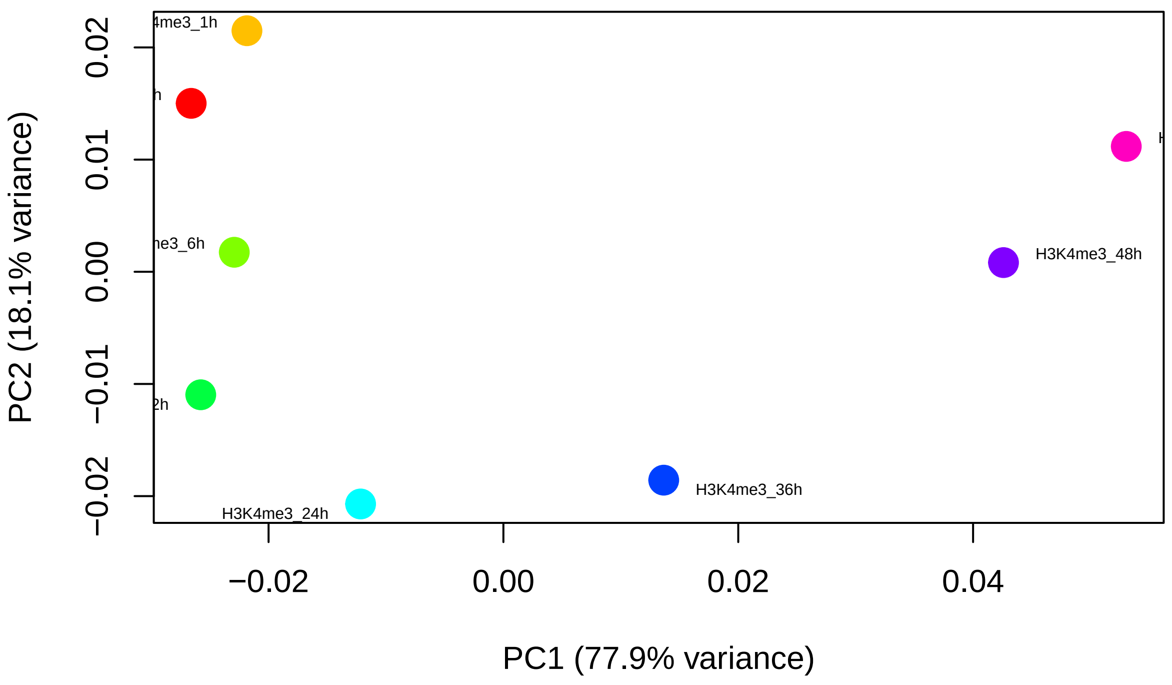
```r
# Using the previous correlation matrix and PCA results
cor.proteome <- cor(Proteome)
pca.proteome <- prcomp(cor.proteome)

# Get group labels and colors
grp <- rownames(pca.proteome$x)  # Set groups according to your data
grp.col <- rainbow(nrow(pca.proteome$x))
names(grp.col) <- rownames(pca.proteome$x)

# Plot the PCA
plot(pca.proteome$x[,1], pca.proteome$x[,2], col=grp.col[grp], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.proteome)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.proteome)$importance[2,2]*100,1), "% variance)"))
```



h3k4me3

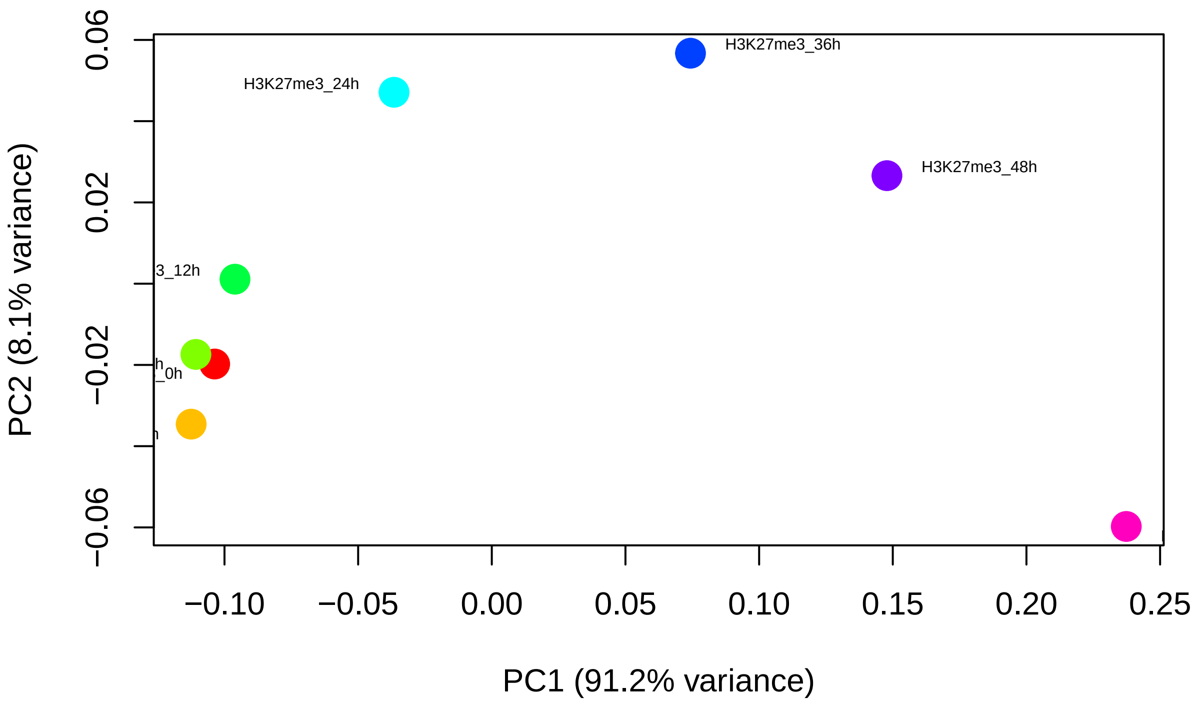
```r
# PCA analysis on the correlation matrix of the H3K4me3 data
cor.h3k4me3 <- cor(H3K4me3)
pca.h3k4me3 <- prcomp(cor.h3k4me3)

# Get group labels and colors
grp <- rownames(pca.h3k4me3$x)
grp.col <- rainbow(nrow(pca.h3k4me3$x))
names(grp.col) <- rownames(pca.h3k4me3$x)

# Generate PCA plot for H3K4me3
plot(pca.h3k4me3$x[,1], pca.h3k4me3$x[,2], col=grp.col[grp], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.h3k4me3)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.h3k4me3)$importance[2,2]*100,1), "% variance)"))

# Add sample labels to the plot
```

```r
calibrate::textxy(pca.h3k4me3$x[,1], pca.h3k4me3$x[,2], labs=grp, cex=0.5)
```



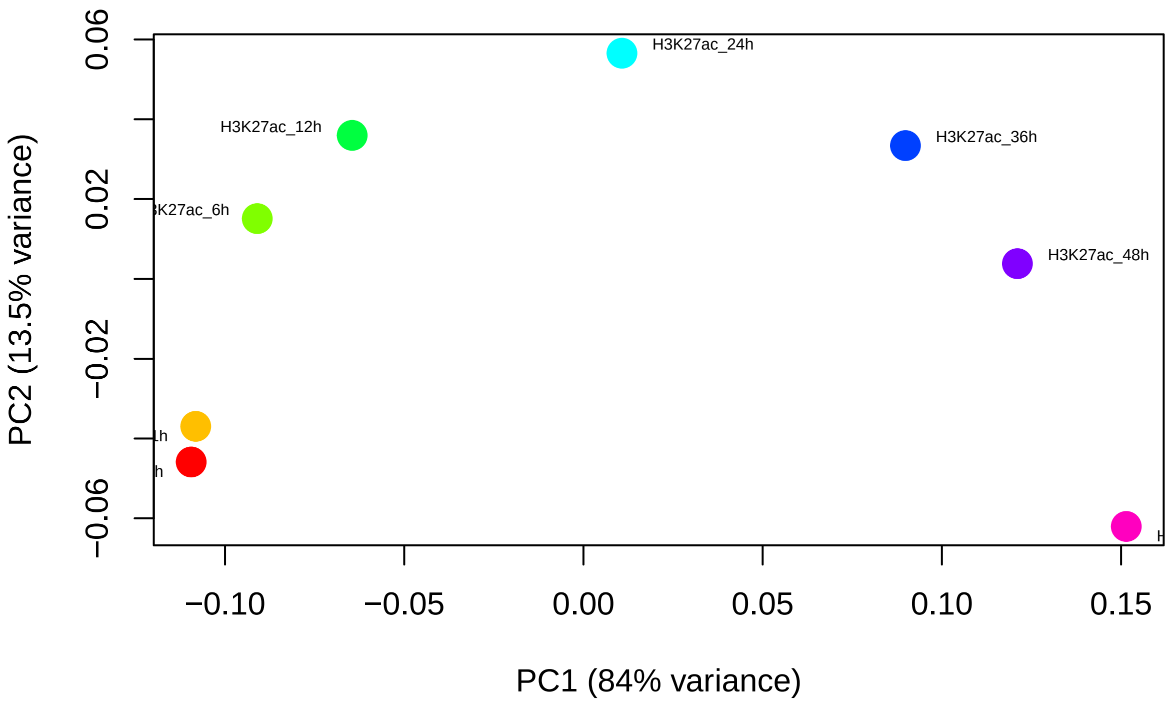
H3K27me3

```r
# PCA analysis for H3K27me3 data
cor.h3k27me3 <- cor(H3K27me3)
pca.h3k27me3 <- prcomp(cor.h3k27me3)

# Update group labels and colors for H3K27me3
grp.h3k27me3 <- rownames(pca.h3k27me3$x)
grp.col.h3k27me3 <- rainbow(nrow(pca.h3k27me3$x))
names(grp.col.h3k27me3) <- rownames(pca.h3k27me3$x)

# Generate PCA plot for H3K27me3
plot(pca.h3k27me3$x[,1], pca.h3k27me3$x[,2], col=grp.col.h3k27me3[grp.h3k27me3], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.h3k27me3)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.h3k27me3)$importance[2,2]*100,1), "% variance)"))

# Correctly label the samples for H3K27me3
calibrate::textxy(pca.h3k27me3$x[,1], pca.h3k27me3$x[,2], labs=grp.h3k27me3, cex=0.5)
```



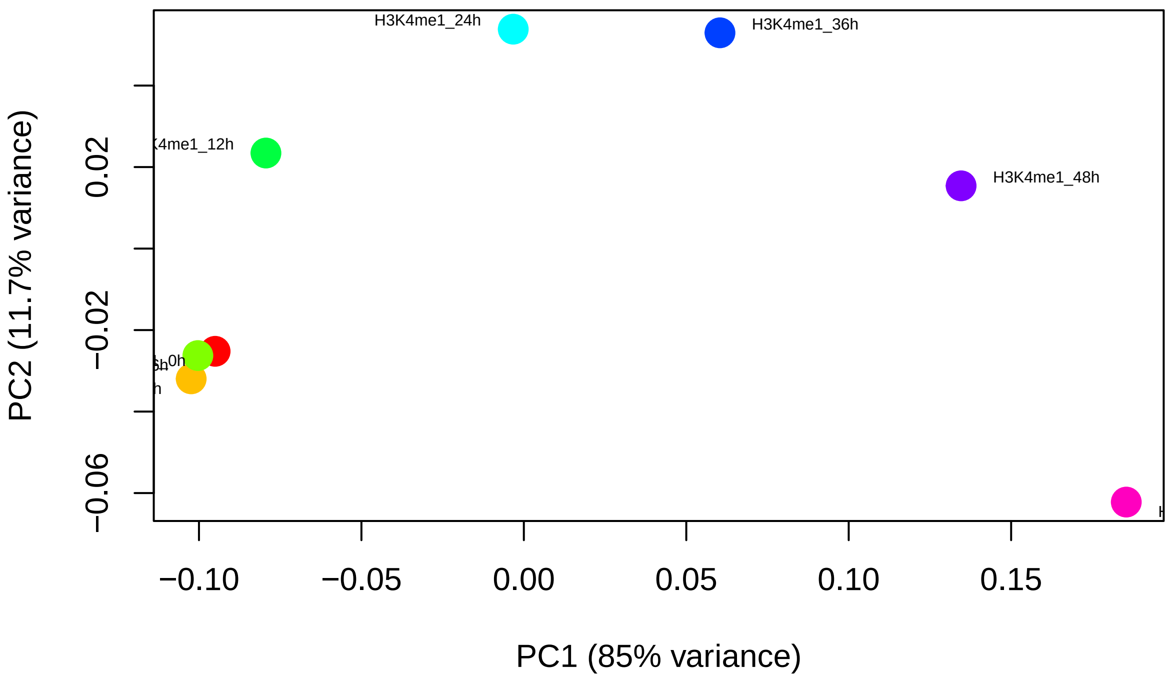
H3K27ac

```r
# PCA analysis for H3K27ac data
cor.h3k27ac <- cor(H3K27ac)
pca.h3k27ac <- prcomp(cor.h3k27ac)

# Update group labels and colors for H3K27ac
grp.h3k27ac <- rownames(pca.h3k27ac$x)
grp.col.h3k27ac <- rainbow(nrow(pca.h3k27ac$x))
names(grp.col.h3k27ac) <- rownames(pca.h3k27ac$x)

# Generate PCA plot for H3K27ac
plot(pca.h3k27ac$x[,1], pca.h3k27ac$x[,2], col=grp.col.h3k27ac[grp.h3k27ac], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.h3k27ac)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.h3k27ac)$importance[2,2]*100,1), "% variance)"))

# Correctly label the samples for H3K27ac
calibrate::textxy(pca.h3k27ac$x[,1], pca.h3k27ac$x[,2], labs=grp.h3k27ac, cex=0.5)
```



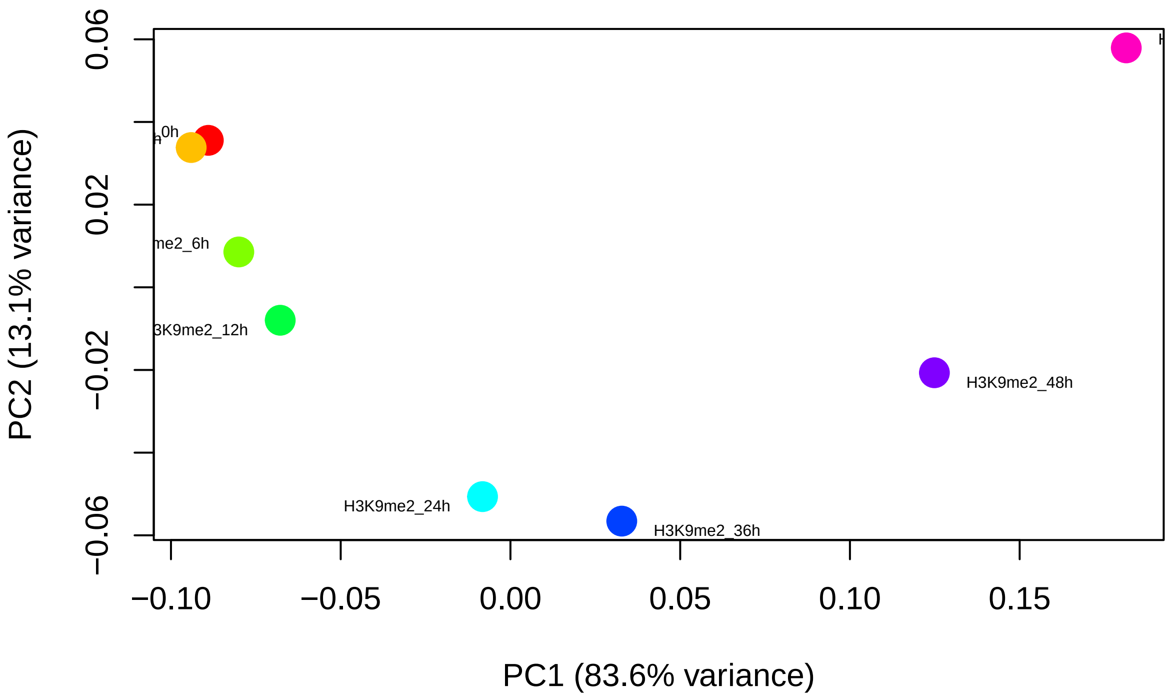
H3K4me1

```
# PCA analysis for H3K4me1 data
cor.h3k4me1 <- cor(H3K4me1)
pca.h3k4me1 <- prcomp(cor.h3k4me1)

# Define the group labels and colors specifically for H3K4me1 data
grp.h3k4me1 <- rownames(pca.h3k4me1$x)
grp.col.h3k4me1 <- rainbow(nrow(pca.h3k4me1$x))
names(grp.col.h3k4me1) <- rownames(pca.h3k4me1$x)

# Generate PCA plot for H3K4me1
plot(pca.h3k4me1$x[,1], pca.h3k4me1$x[,2], col=grp.col.h3k4me1[grp.h3k4me1], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.h3k4me1)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.h3k4me1)$importance[2,2]*100,1), "% variance)"))

# Correctly label the samples for H3K4me1
calibrate::textxy(pca.h3k4me1$x[,1], pca.h3k4me1$x[,2], labs=grp.h3k4me1, cex=0.5)
```



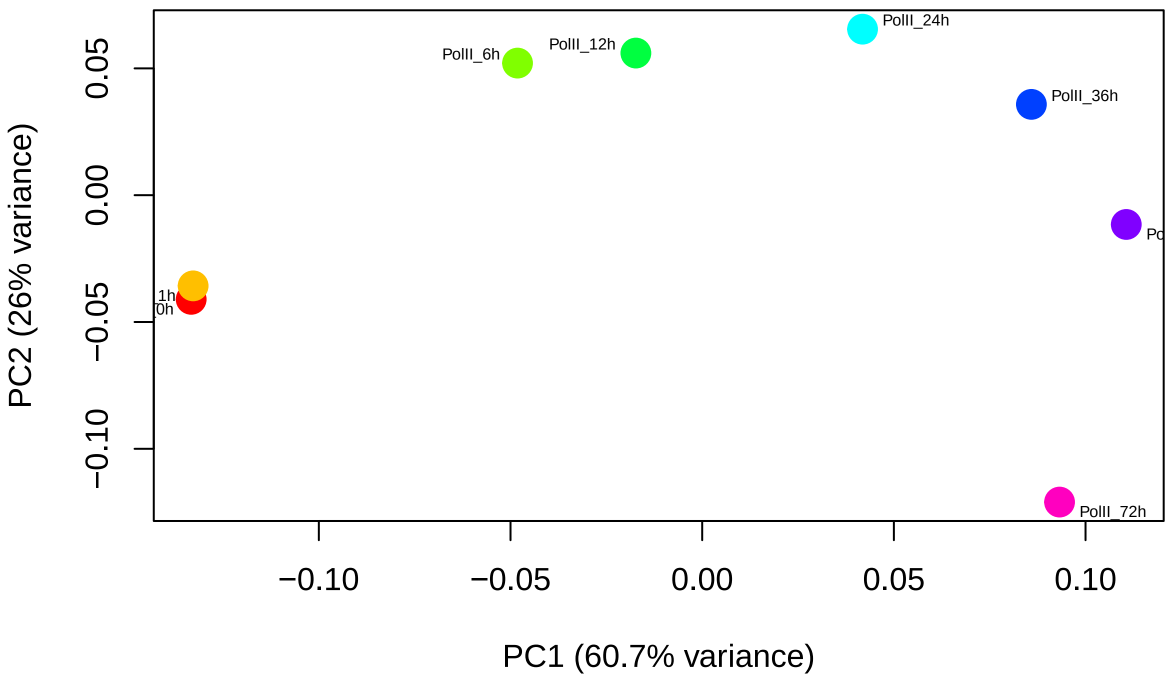
H3K9me2

```
# PCA analysis for H3K9me2 data
cor.h3k9me2 <- cor(H3K9me2)
pca.h3k9me2 <- prcomp(cor.h3k9me2)

# Define the group labels and colors specifically for H3K9me2 data
grp.h3k9me2 <- rownames(pca.h3k9me2$x)
grp.col.h3k9me2 <- rainbow(nrow(pca.h3k9me2$x))
names(grp.col.h3k9me2) <- rownames(pca.h3k9me2$x)

# Generate PCA plot for H3K9me2
plot(pca.h3k9me2$x[,1], pca.h3k9me2$x[,2], col=grp.col.h3k9me2[grp.h3k9me2], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.h3k9me2)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.h3k9me2)$importance[2,2]*100,1), "% variance)"))

# Correctly label the samples for H3K9me2
calibrate::textxy(pca.h3k9me2$x[,1], pca.h3k9me2$x[,2], labs=grp.h3k9me2, cex=0.5)
```



PolII

```
# PCA analysis for PolII data
cor.polii <- cor(PolII)
pca.polii <- prcomp(cor.polii)

# Define the group labels and colors specifically for PolII data
grp.polii <- rownames(pca.polii$x)
grp.col.polii <- rainbow(nrow(pca.polii$x))
names(grp.col.polii) <- rownames(pca.polii$x)

# Generate PCA plot for PolII
plot(pca.polii$x[,1], pca.polii$x[,2], col=grp.col.polii[grp.polii], pch=19, cex=2,
     xlab=paste0("PC1 (", round(summary(pca.polii)$importance[2,1]*100,1), "% variance)"),
     ylab=paste0("PC2 (", round(summary(pca.polii)$importance[2,2]*100,1), "% variance)"))

# Correctly label the samples for PolII
calibrate::textxy(pca.polii$x[,1], pca.polii$x[,2], labs=grp.polii, cex=0.5)
```



Classification

Filter and Combine Datasets

In order to properly model and predicting novel transcription factors target genes we join the 8 datasets together and perform our calculations on the larger dataset.

To ensure consistency across datasets, we filter each dataset to include only the common genes present in all omics layers. We then combine these filtered datasets into a single data frame for further analysis.

```
# Verify all data has the same set of genes
genes <- intersect(rownames(Transcriptome), rownames(Proteome))
genes <- intersect(genes, rownames(H3K4me3))
genes <- intersect(genes, rownames(H3K27me3))
genes <- intersect(genes, rownames(H3K27ac))
genes <- intersect(genes, rownames(H3K4me1))
genes <- intersect(genes, rownames(H3K9me2))
genes <- intersect(genes, rownames(PolII))
```

```
# Filter each dataset for the common genes
Transcriptome_filter <- Transcriptome[genes, ]
Proteome_filter <- Proteome[genes, ]
H3K4me3_filter <- H3K4me3[genes, ]
H3K27me3_filter <- H3K27me3[genes, ]
H3K27ac_filter <- H3K27ac[genes, ]
H3K4me1_filter <- H3K4me1[genes, ]
H3K9me2_filter <- H3K9me2[genes, ]
PolII_filter <- PolII[genes, ]

# Confirm that all datasets share the same gene
identical(rownames(Transcriptome_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
identical(rownames(Proteome_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
identical(rownames(H3K27ac_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
identical(rownames(H3K4me1_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
identical(rownames(H3K9me2_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
identical(rownames(PolII_filter), rownames(H3K4me3_filter))
```

```
## [1] TRUE
```

```r
# Rename columns to avoid conflicts
colnames(Transcriptome_filter) <- paste("T_", colnames(Transcriptome_filter), sep = "")
colnames(Proteome_filter) <- paste("P_", colnames(Proteome_filter), sep = "")
colnames(H3K4me3_filter) <- paste("H3K4me3_", colnames(H3K4me3_filter), sep = "")
colnames(H3K27me3_filter) <- paste("H3K27me3_", colnames(H3K27me3_filter), sep = "")
colnames(H3K27ac_filter) <- paste("H3K27ac_", colnames(H3K27ac_filter), sep = "")
colnames(H3K4me1_filter) <- paste("H3K4me1_", colnames(H3K4me1_filter), sep = "")
colnames(H3K9me2_filter) <- paste("H3K9me2_", colnames(H3K9me2_filter), sep = "")
colnames(PolII_filter) <- paste("PolII_", colnames(PolII_filter), sep = "")

# Combine the datasets
combined_data <- cbind(
  Transcriptome_filter,
  Proteome_filter,
  H3K4me3_filter,
  H3K27me3_filter,
  H3K27ac_filter,
  H3K4me1_filter,
  H3K9me2_filter,
  PolII_filter
)

# Add the labels
label <- ifelse(genes %in% OSN_target_genes_subset, "OSN", "Other")
combined_data <- data.frame(combined_data)
combined_data$label <- factor(label)
```

```r
# Number of genes which are known to be targets for Sox2 and Nanog
length(OSN_target_genes_subset)
```

```
## [1] 100
```

We have 100 known target genes for OSN, and as seen below the we have 95 genes that have been identified as novel Sox2/Nanog targets on our combined filtered dataset.

```r
# Check the initial label distribution
print(table(combined_data$label))
```

```
##
```

```
##   OSN Other
##    95  8085
```

Data Splitting and Balancing

The dataset is split into training (90%) and testing (10%) sets. The label column is reassigned to the test set to confirm that it is included correctly.

```
# Split the dataset into training (90%) and testing (10%) sets
set.seed(123)
train_index <- createDataPartition(combined_data$label, p = 0.9, list = FALSE)

train_data <- combined_data[train_index, ]
test_data <- combined_data[-train_index, ]

# Reassign the label column to test_data
test_data$label <- combined_data[-train_index, "label"]

# Check the distribution of labels in the training and test sets
print("Training set label distribution:")
```

```
## [1] "Training set label distribution:"
```

```
print(table(train_data$label))
```

```
##
##   OSN Other
##    86  7277
```

```
print("Test set label distribution:")
```

```
## [1] "Test set label distribution:"
```

```
print(table(test_data$label))
```

```
##
##   OSN Other
##     9   808
```

Balancing the Training Data

To address the imbalance in the dataset, downsampling is used on both classes, `OSN` and `Other`, to make sure they are represented equally. This technique improves the model's accuracy and generalization by preventing bias towards the more frequent class.

```
# Balance the training data using downsampling
downsampled_train_data <- downSample(x = train_data[, -ncol(train_data)],
                                     y = train_data$label,
                                     list = FALSE, yname = "label")

# Display the new balanced label distribution
print("Balanced training set label distribution:")
```

```
## [1] "Balanced training set label distribution:"
```

```
table(downsampled_train_data$label)
```

```
##
##   OSN Other
##     86    86
```

```r
# Display dimensions of the balanced training data
dim(downsampled_train_data)
```

```
## [1] 172  64
```

```r
# Final check of training dataset dimensions
print(dim(downsampled_train_data))
```

```
## [1] 172  64
```

Model Training

Two models, SVM with a radial kernel and Random Forest, are trained using the balanced dataset to compare their performance under balanced class distribution conditions.

```r
# Train an SVM model on the downsampled training data with radial basis function kernel
svm_model <- svm(label ~ ., data = downsampled_train_data, kernel = "radial", probability = TRUE)

# Train a Random Forest model
rf_model <- randomForest(label ~ ., data = downsampled_train_data, ntree = 1000)

# Extract feature importance and order it by decreasing importance
importance <- importance(rf_model)

ord <- order(importance, decreasing = TRUE)

# Select only the top 10 most important features
top_features <- names(importance)[ord][1:10]
top_importance_values <- importance[ord][1:10]

# Check that top features are correctly identified
if (length(top_features) < 10) {
    warning("Less than 10 features are available. Adjusting to available number.")
    top_features <- names(importance)[ord]
    top_importance_values <- importance[ord]
}
```

```
## Warning: Less than 10 features are available. Adjusting to available number.
```

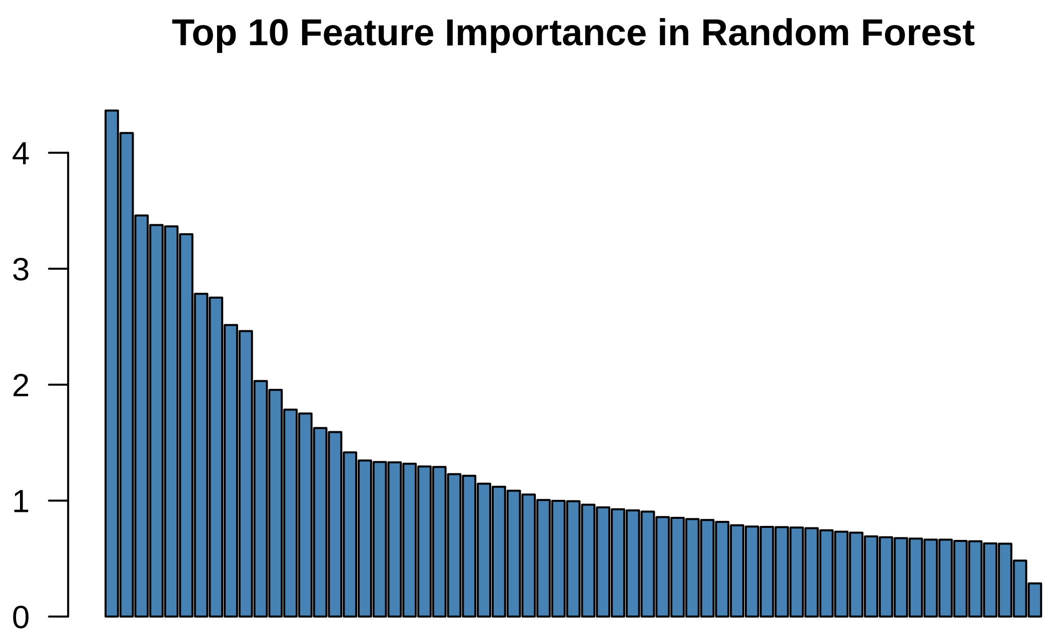
```r
# Print the top 10 features
cat("Top 10 Most Important Features:\n")
```

```
## Top 10 Most Important Features:
```

```r
print(top_features)
```

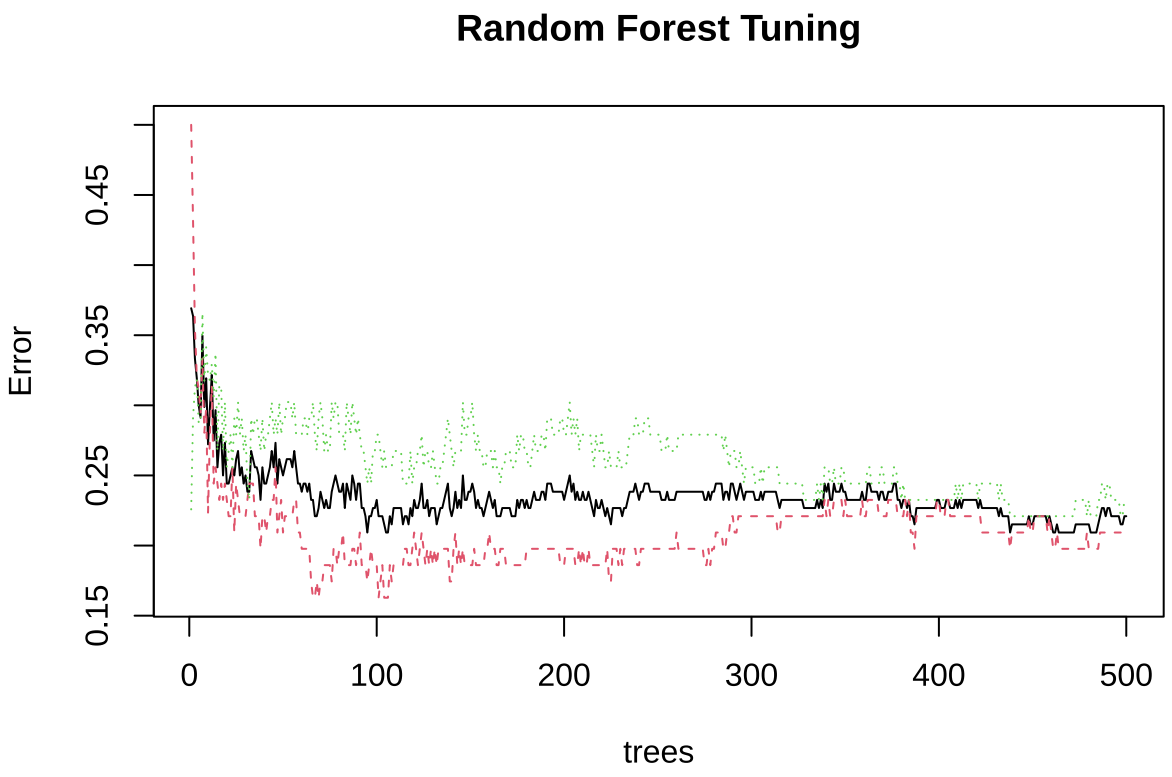
```
## NULL
```

```r
# Plot the importance of the top 10 features
barplot(top_importance_values, names.arg = top_features,
        main = "Top 10 Feature Importance in Random Forest",
        col = 'steelblue',
        las = 2,
        cex.names = 0.7)
```



We can then perform cross-validation and finetune the hyperparameters, being careful not to overfit the model.

```r
# Define tuning grid and control setup
tuning_grid <- expand.grid(mtry = seq(2, 5, by = 1))
control <- trainControl(method = "cv", number = 5)

# Tuning the model
set.seed(123)
tuned_rf_model <- train(label ~ ., data = downsampled_train_data,
                        method = "rf", trControl = control, tuneGrid = tuning_grid)

# Plotting tuning results
plot(tuned_rf_model$finalModel, main="Random Forest Tuning")
```



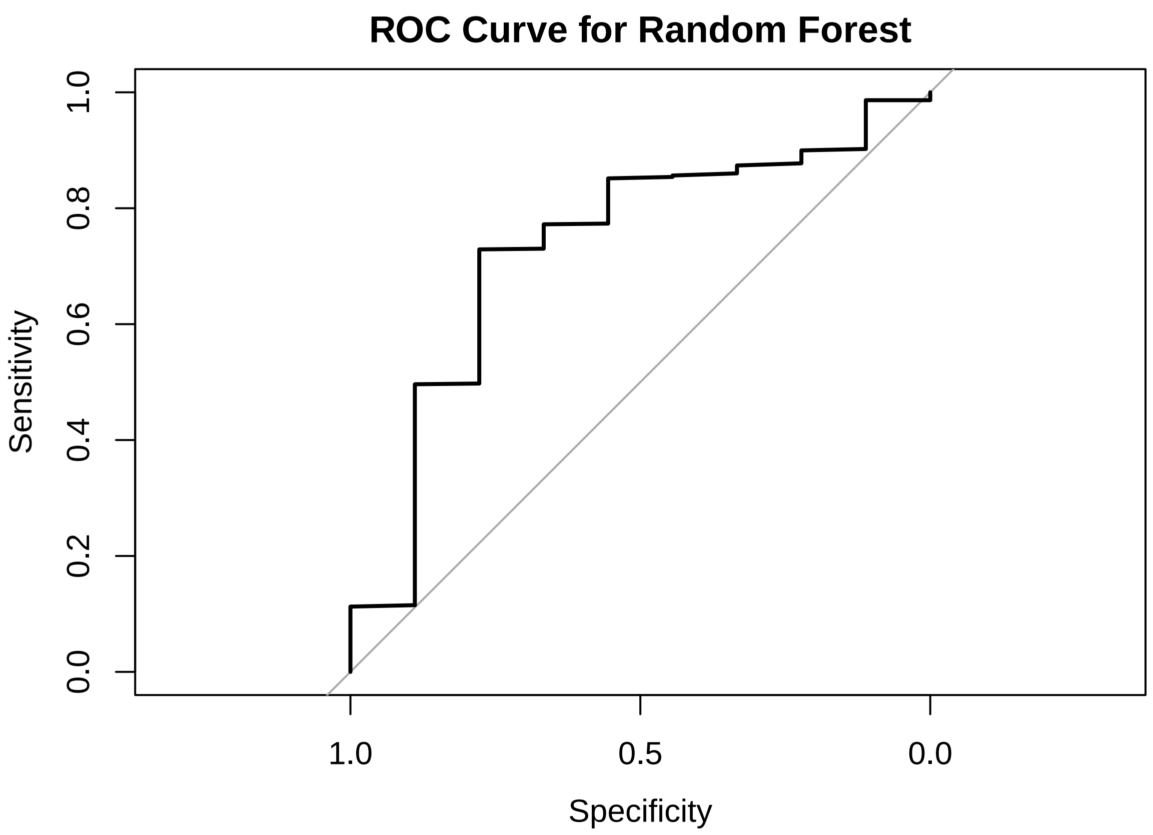
Before conducting proper evaluation later in this report, we can briefly evaluate the performance of this model.

```
rf_predictions <- predict(tuned_rf_model, newdata = test_data, type = "prob")
roc_curve <- roc(response = test_data$label, predictor = as.numeric(rf_predictions[,2]))
```

```
## Setting levels: control = OSN, case = Other
```

```
## Setting direction: controls < cases
```

```
# Plot ROC curve
plot(roc_curve, main = "ROC Curve for Random Forest")
```



Upon viewing the performance of the model above, it was clear that more could be done to optimise the model. As a result, the team decided to experiment with other models to evaluate the effectiveness of variations of standard random forest models.

Bagging w/ Bagged Trees

Bagging improves stability and accuracy by reducing variance and avoiding overfitting.

```
bagged_trees <- train(
  label ~ .,
  data = downsampled_train_data,
  method = "treebag",
  trControl = trainControl(method = "cv", number = 5),
  tuneLength = 5
)
print(bagged_trees)
```

```
## Bagged CART
##
## 172 samples
##  63 predictor
##   2 classes: 'OSN', 'Other'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 138, 137, 138, 138, 137
## Resampling results:
##
##   Accuracy   Kappa
##   0.7440336  0.4883745
```

Gradient Boosting with Hyperparameter Tuning Using xgboost (using parallel processing)

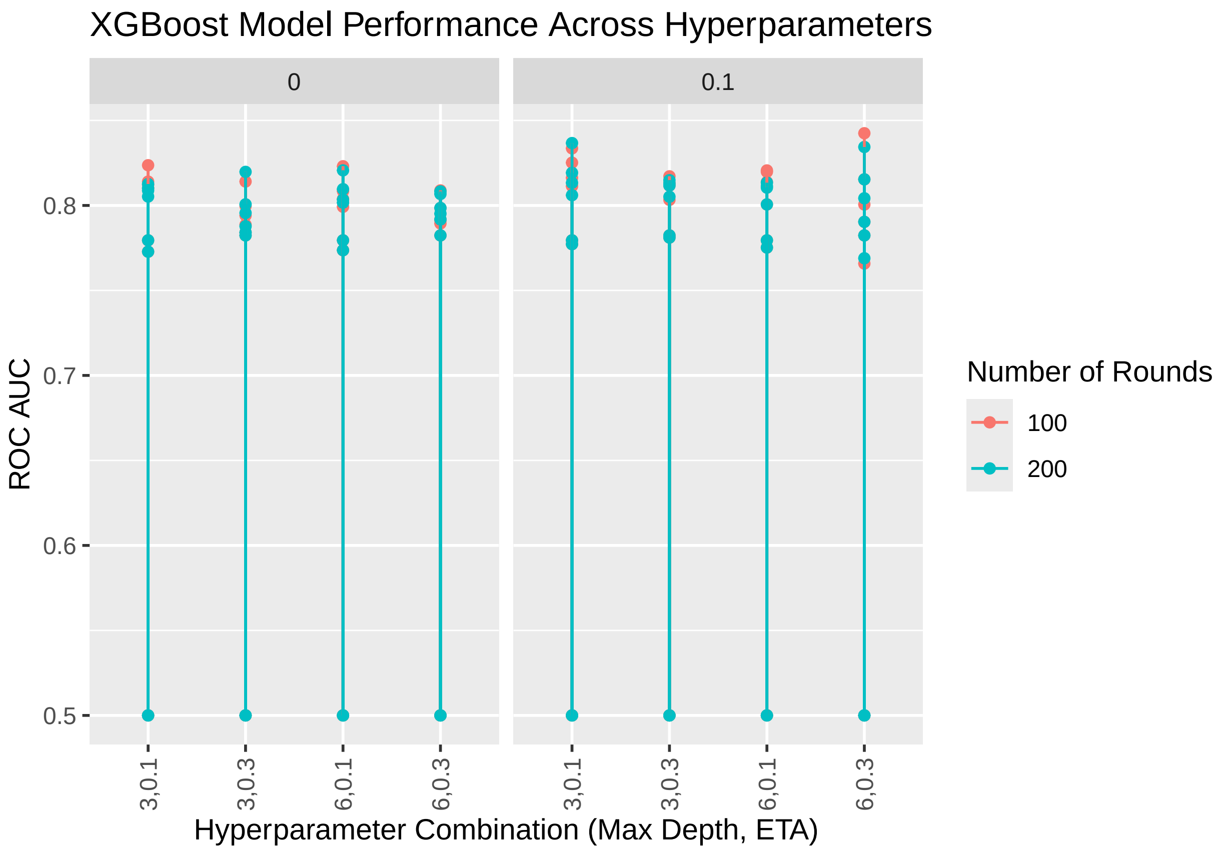
```r
# Set up a tuning grid for xgboost
tune_grid_xgb <- expand.grid(
  nrounds = c(100, 200),
  max_depth = c(3, 6),
  eta = c(0.1, 0.3),
  gamma = c(0, 0.1),
  colsample_bytree = c(0.5, 1),
  min_child_weight = c(1, 10),
  subsample = c(0.5, 1)
)

# Enable parallel processing
cl <- makeCluster(detectCores())
registerDoParallel(cl)

# Define control function with ROC as the metric
train_control <- trainControl(
  method = "cv",
  number = 3,
  savePredictions = "final",
  verboseIter = TRUE,
  allowParallel = TRUE,
  summaryFunction = twoClassSummary,  # Computes ROC and Sensitivity among others
  classProbs = TRUE,  # Needed for ROC and AUC calculations
  selectionFunction = "best"  # Chooses the best tuning parameters based on ROC
)


# Train xgboost model with tuning
xgb_model_tuned <- train(
  label ~ .,
  data = downsampled_train_data,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = tune_grid_xgb,
  metric = "ROC"
)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting nrounds = 100, max_depth = 6, eta = 0.3, gamma = 0.1, colsample_bytree = 0.5, min_child_weig
```

```r
# Stop and unregister parallel processing
stopCluster(cl)
registerDoSEQ()

# Print the best tuning parameters
print(xgb_model_tuned$bestTune)
```

```
##     nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 113     100         6 0.3   0.1              0.5                1       0.5
```

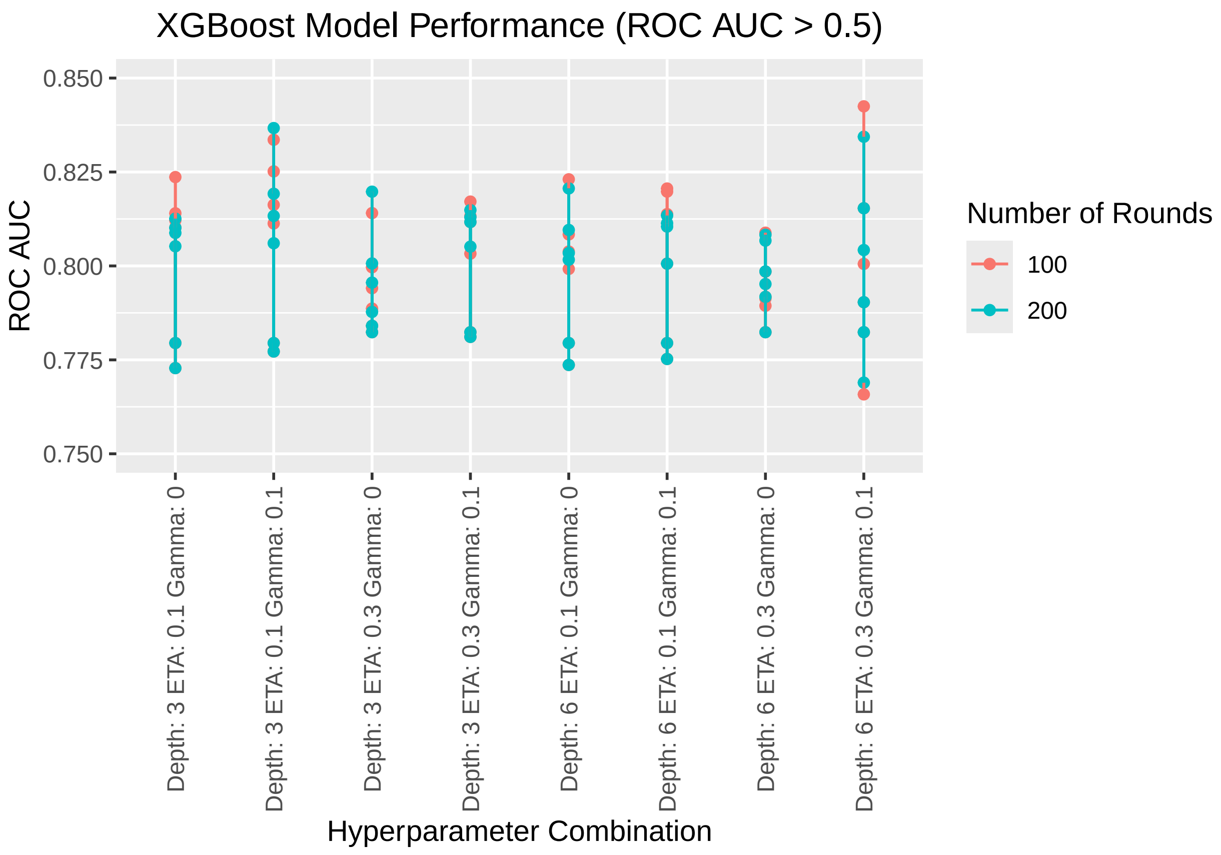
```r
# Prepare data for plotting
results <- xgb_model_tuned$results
results$interaction <- interaction(results$max_depth, results$eta, results$gamma,
                                   results$colsample_bytree,
                                   results$min_child_weight,
                                   results$subsample, drop = TRUE)

results$interaction <- factor(results$interaction)
results$simplified_interaction <- with(results, paste(max_depth, eta, sep=","))

ggplot(data = results, aes(x = simplified_interaction, y = ROC)) +
  geom_point(aes(color = as.factor(nrounds))) +
  geom_line(aes(group = simplified_interaction, color = as.factor(nrounds))) +
  facet_wrap(~ gamma, scales = "free_x") +
  labs(title = "XGBoost Model Performance Across Hyperparameters",
       x = "Hyperparameter Combination (Max Depth, ETA)",
       y = "ROC AUC",
       color = "Number of Rounds") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



As this graph isn't very useful, especially given the varying performance from various hypertuned models, we can make this clearer by removing all outputs below 0.5 to view the performance of the more successful models.

```r
# Filter the data to include only ROC AUC values above 0.5
filtered_results <- results[results$ROC > 0.5,]
```

```r
# Create a readable label for the x-axis (simplifying or detailing as needed)
filtered_results$readable_interaction <- with(filtered_results, paste("Depth:", max_depth, "ETA:", eta,

# Plot the performance focusing on ROC AUC values above 0.5
library(ggplot2)
ggplot(data = filtered_results, aes(x = readable_interaction, y = ROC)) +
  geom_point(aes(color = as.factor(nrounds))) +
  geom_line(aes(group = readable_interaction, color = as.factor(nrounds))) +
  labs(title = "XGBoost Model Performance (ROC AUC > 0.5)",
       x = "Hyperparameter Combination",
       y = "ROC AUC",
       color = "Number of Rounds") +
  scale_y_continuous(limits = c(0.75, 0.85), breaks = seq(0.5, 1, by = 0.025)) +  # Setting y-axis to s
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        plot.title = element_text(hjust = 0.5))  # Centering the plot title
```



It is notable that for all of the objectively 'high-performing' models (those with an ROC AUC greater than 0.5), there wasn't much fluctuation beyond a score of ± 0.05 about 0.8, or a 6.25% variation. Therefore, the hypothesis that hypertuning the model would improve its performance does not hold, as the cost of generating such a model does not warrant the marginal increase in performance.

Extended Model Training - Neural Networks

To extend our understanding of the dataset and compare our Random Forest models against other classifiers, the team decided to evaluate the use of a neural network on the same training data.

Preparing Data and Feature Scaling

Proper data scaling is necessary for the performance of a Neural Network due to the sensitivity of the implementation of the algorithm in R to the scale of input variables.

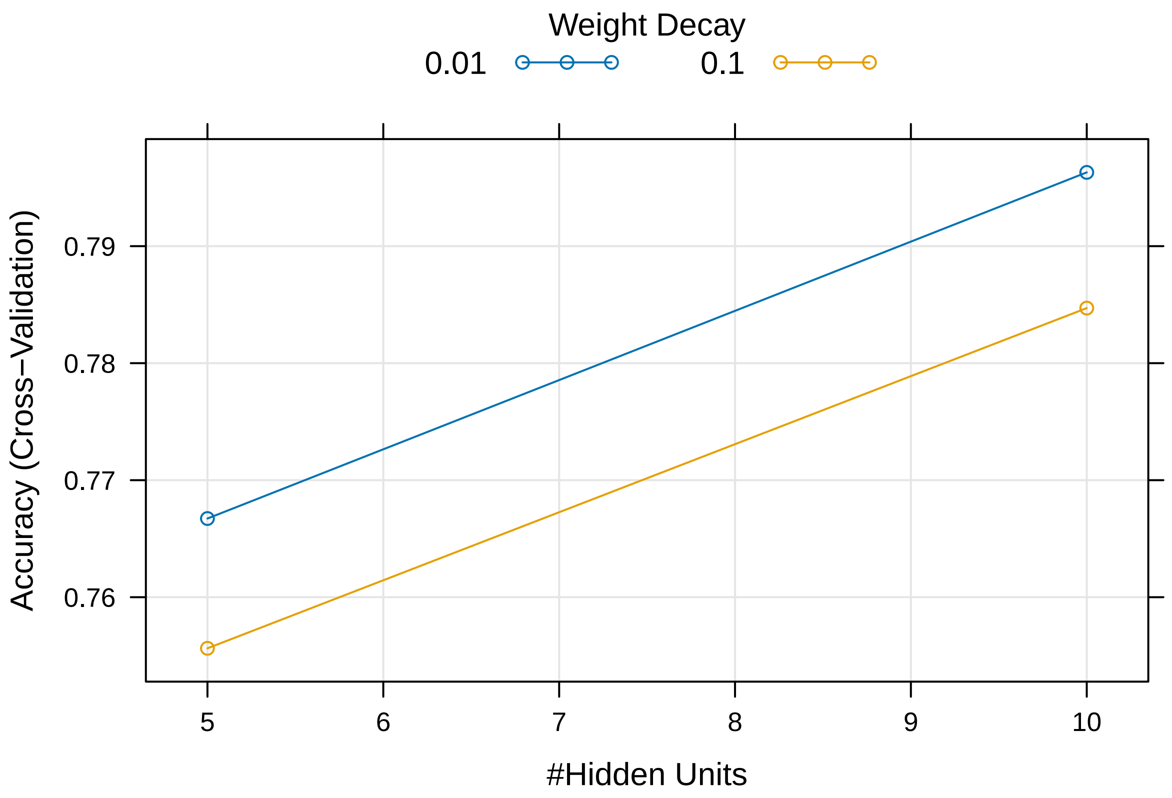
```r
scaled_data <- scale(downsampled_train_data[, -ncol(downsampled_train_data)])
scaled_train_data <- data.frame(scaled_data, label = downsampled_train_data$label)
```

Then, as was done with the Random Forest model, the team trained the Neural Network with varying architectural parameters and visualizing the tuning process to identify the best model settings.

```r
# Setup for Neural Network training
control_nn <- trainControl(method = "cv", number = 5, savePredictions = "final")
grid_nn <- expand.grid(.size = c(5, 10), .decay = c(0.1, 0.01))

# Train the Neural Network
set.seed(123)
nn_model <- train(label ~ ., data = scaled_train_data, method = "nnet", trControl = control_nn, tuneGri
```

We can then plot the model's performance.

```r
plot(nn_model)
```



Model Evaluation

Predictions and Confusion Matrix We use the trained models to make predictions on the test set and evaluate their performance using confusion matrices.

```r
# Column names in the test set match the training data
colnames(test_data) <- colnames(downsampled_train_data)[1:(ncol(downsampled_train_data) - 1)]  # Exclud

# Check if the label column is present and correctly populated
```

```
if ("label" %in% colnames(combined_data) && length(combined_data[-train_index, "label"]) == nrow(test_da
    # Assign the label to test_data
    test_data$label <- combined_data[-train_index, "label"]
} else {
    stop("The label vector is empty or has a different length than expected. Check the data preparation
}

# The label is a factor with the correct levels
test_data$label <- factor(test_data$label, levels = c("OSN", "Other"))

# SVM Predictions on the test set
svm_test_pred <- predict(svm_model, newdata = test_data[, -ncol(test_data)], probability = TRUE)
svm_test_prob <- attr(svm_test_pred, "probabilities")[, "OSN"]

# Random Forest Predictions on the test set
rf_test_pred <- predict(rf_model, newdata = test_data[, -ncol(test_data)], type = "prob")[, "OSN"]

# SVM Confusion Matrix on the test set
svm_test_conf_matrix <- table(Predicted = ifelse(svm_test_prob > 0.5, "OSN", "Other"), Actual = test_dat
print("SVM Test Confusion Matrix:")
```

```
## [1] "SVM Test Confusion Matrix:"
```

```
print(svm_test_conf_matrix)
```

```
##          Actual
## Predicted OSN Other
##     OSN     5   190
##     Other   4   618
```

```
# Random Forest Confusion Matrix on the test set
rf_test_conf_matrix <- table(Predicted = ifelse(rf_test_pred > 0.5, "OSN", "Other"), Actual = test_data$
print("Random Forest Test Confusion Matrix:")
```

```
## [1] "Random Forest Test Confusion Matrix:"
```

```
print(rf_test_conf_matrix)
```

```
##          Actual
## Predicted OSN Other
##     OSN     6   216
##     Other   3   592
```

ROC Curve and AUC

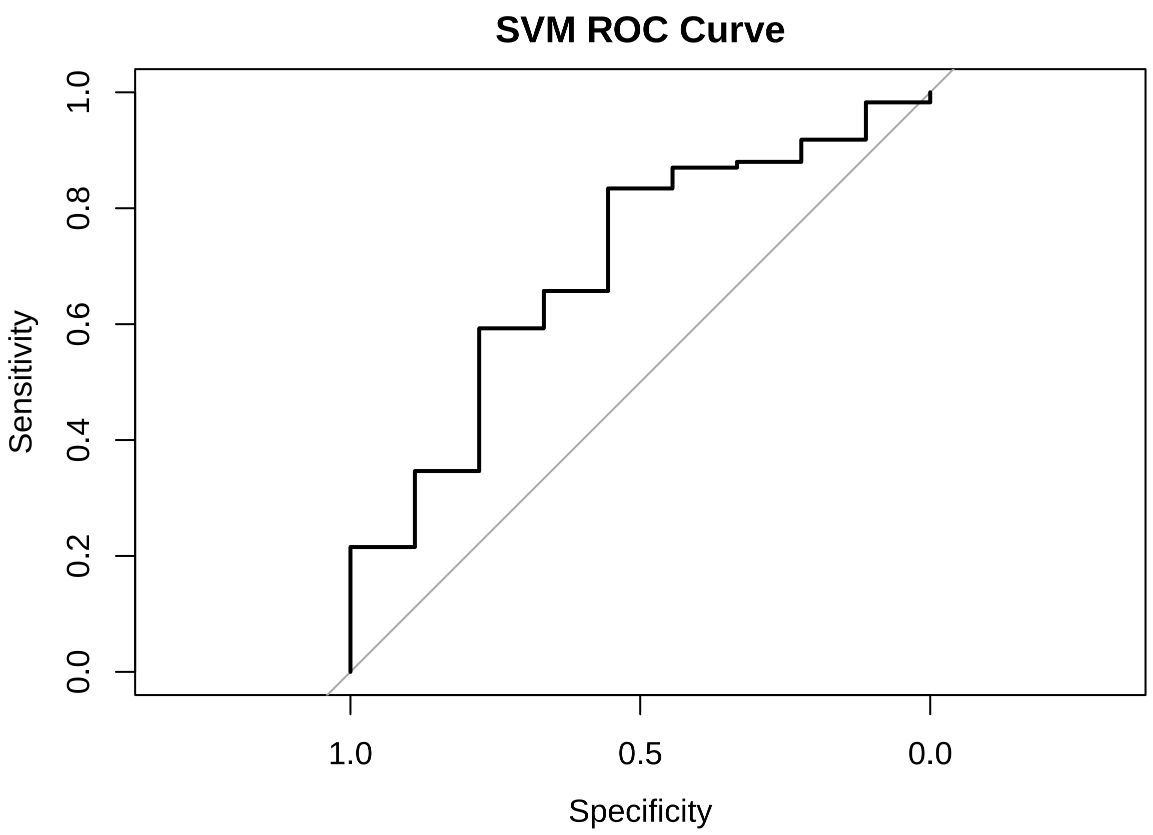
To further assess model performance, we plot the ROC curves and calculate the AUC for both the SVM and Random Forest models.

```
# Evaluate the models on the test set (AUC and ROC)
svm_test_roc <- roc(test_data$label, svm_test_prob)
```

```
## Setting levels: control = OSN, case = Other
```

```
## Setting direction: controls > cases
```

```
plot(svm_test_roc, main = "SVM ROC Curve")
```



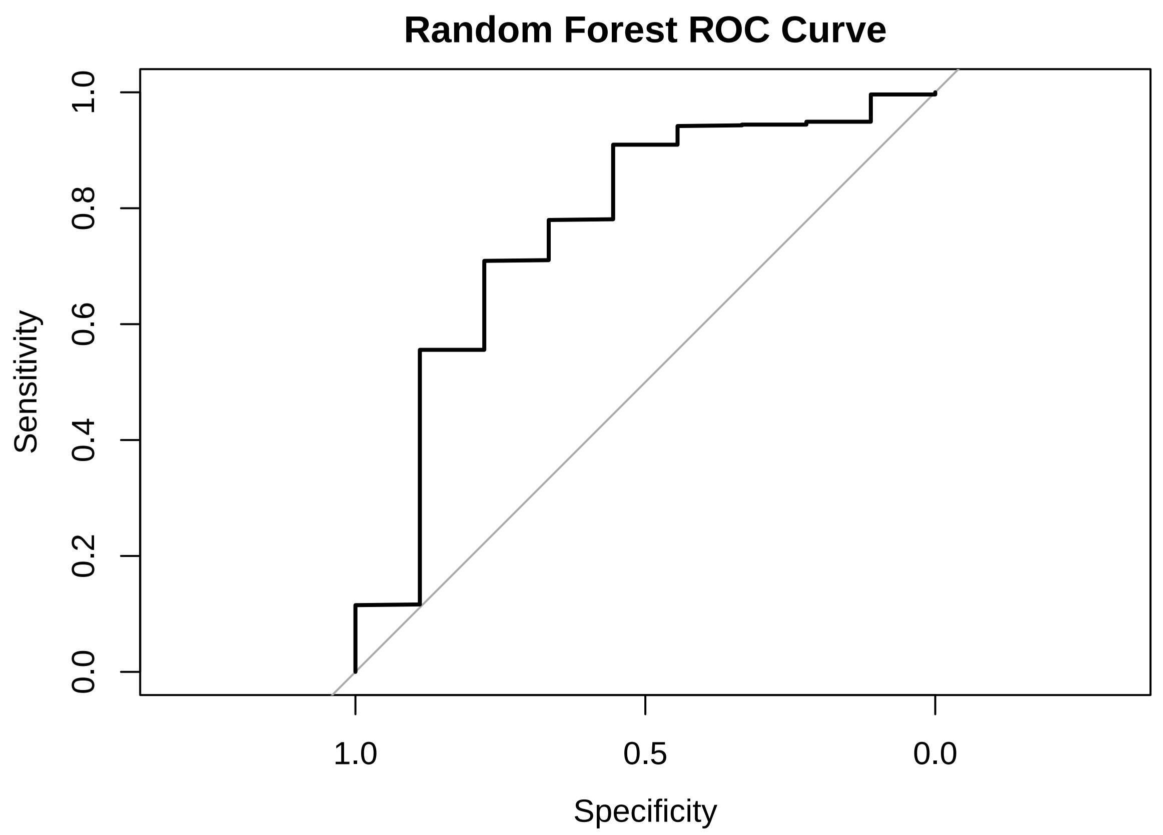
```
print(paste("Final SVM Test AUC:", auc(svm_test_roc)))
```

```
## [1] "Final SVM Test AUC: 0.6996699669967"
```

```
rf_test_roc <- roc(test_data$label, rf_test_pred)
```

```
## Setting levels: control = OSN, case = Other
## Setting direction: controls > cases
```

```
plot(rf_test_roc, main = "Random Forest ROC Curve")
```



```r
print(paste("Final Random Forest Test AUC:", auc(rf_test_roc)))
```

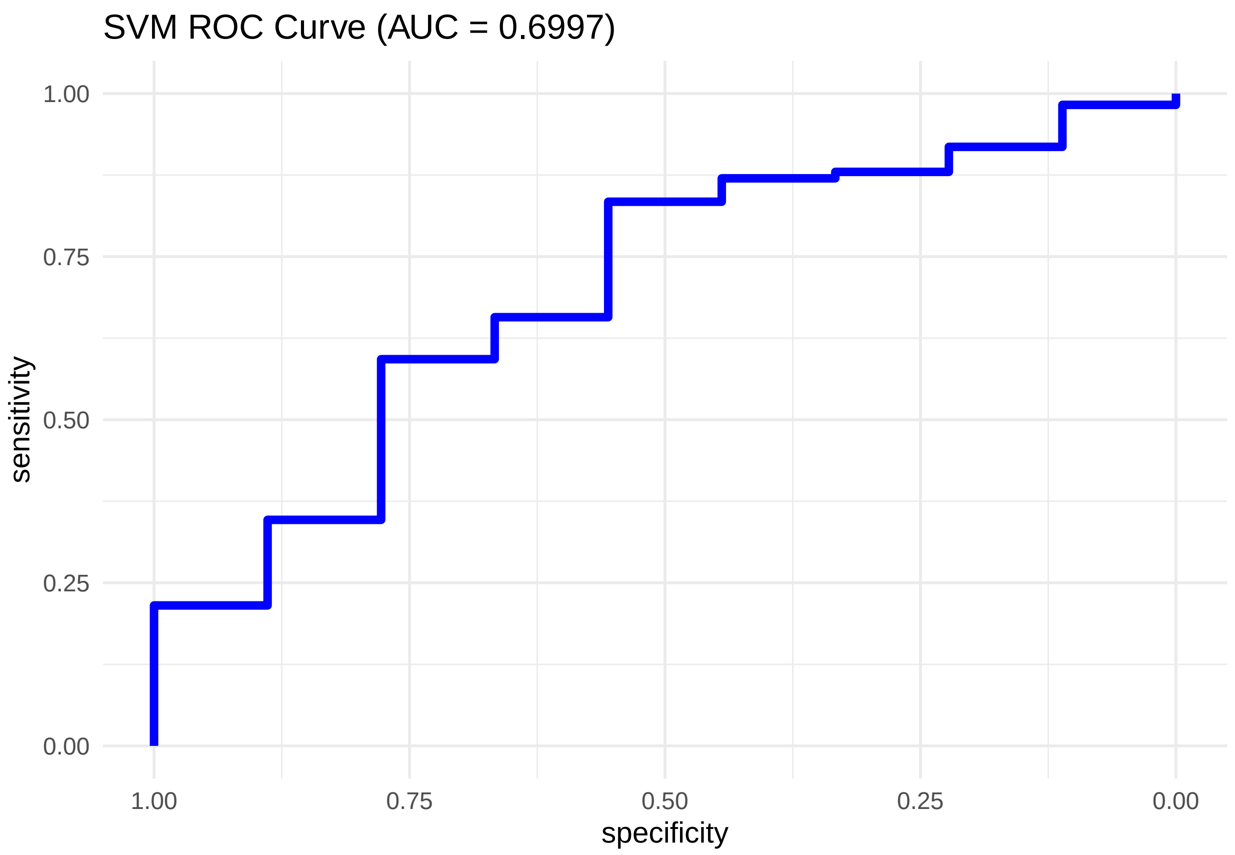
```
## [1] "Final Random Forest Test AUC: 0.767051705170517"
```

```r
# Evaluate SVM model on the test set (AUC and ROC)
svm_test_roc <- roc(test_data$label, svm_test_prob)
```

```
## Setting levels: control = OSN, case = Other
```

```
## Setting direction: controls > cases
```

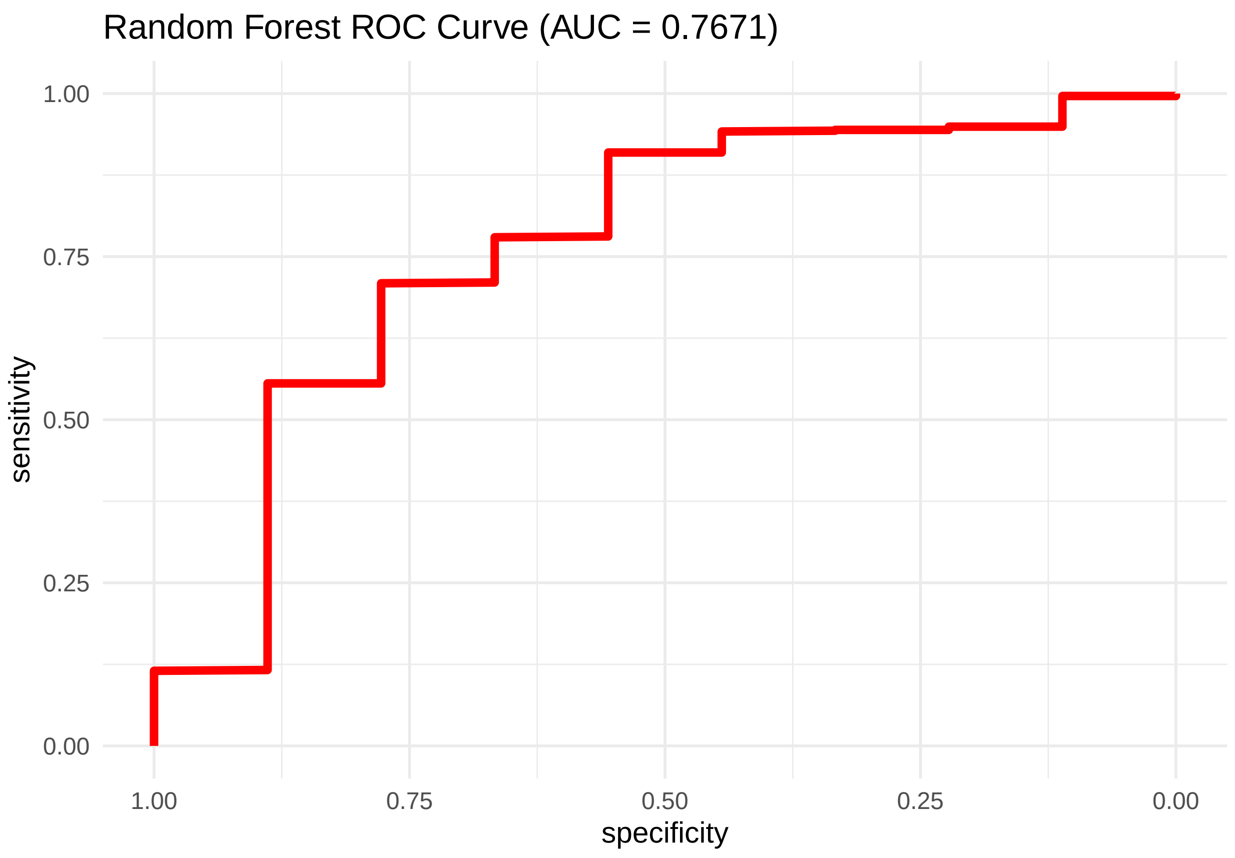
```r
svm_test_auc <- round(auc(svm_test_roc), 4)

# Plot ROC curve for SVM
ggroc(svm_test_roc, colour = 'blue', size = 1.5) +
  ggtitle(paste0('SVM ROC Curve (AUC = ', svm_test_auc, ')')) +
  theme_minimal()
```



```r
# Evaluate Random Forest model on the test set (AUC and ROC)
rf_test_roc <- roc(test_data$label, rf_test_pred)
```

```
## Setting levels: control = OSN, case = Other
## Setting direction: controls > cases
```

```r
rf_test_auc <- round(auc(rf_test_roc), 4)

# Plot ROC curve for Random Forest
ggroc(rf_test_roc, colour = 'red', size = 1.5) +
  ggtitle(paste0('Random Forest ROC Curve (AUC = ', rf_test_auc, ')')) +
  theme_minimal()
```



Future Directions:

Model Improvement:
Further hyperparameter tuning using automated methods like grid search or random search could refine the model's accuracy. Additionally, exploring ensemble methods that combine predictions from several models might yield better results.

Data Expansion:
Incorporating additional omics layers, such as metabolomics or additional transcription factor binding profiles, could help to improve the model's predictive power and generalisability.

Integration with Clinical Data:
Linking omics profiles with clinical outcomes could also be explored to improve the translational impact of the research.