STAT430 Assignment 3.

Tree-based Methods | SVMs

Ben Murarotto

May 2025

Contents

1 Chatter dataset.

The activities of the instant chat function on a company website were reviewed to identify key features that allowed a given query to be *successfully resolved*. The dataset Chatter.csv contains the following variables:

Worker: 10 point self-assessment score on session given by the employee *Max_msg*: Maximum number of characters used in any message *Min_msg*: Minimum number of characters used in any message *Exchanges*: The total number of messages exchanged during the session *Total_time*: Total time the customer was active for *Time_length*: Average time (in secs) customer waited for a response from the employee *Age_client*: Age of the customer *Resolved*: Whether the customer considered the issue resolved ("No" or "Yes")

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.4.3
```

```
library(ggplot2)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.4.3
```

```
chatter.df <- read.csv("Chatter.csv", header = T)
summary(chatter.df)
```

```
##      Worker        Max_msg         Min_msg         Exchanges
##  Min.   :1.00   Min.   :  0.0   Min.   :  0.00   Min.   : 1.00
##  1st Qu.:2.00   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 2.00
##  Median :3.00   Median :115.0   Median : 72.00   Median :12.00
##  Mean   :3.94   Mean   :119.6   Mean   : 68.99   Mean   :11.18
##  3rd Qu.:6.00   3rd Qu.:140.0   3rd Qu.: 80.00   3rd Qu.:17.00
##  Max.   :9.00   Max.   :199.0   Max.   :122.00   Max.   :50.00
##    Total_time     Time_length       Age_client      Resolved
##  Min.   : 0.00   Min.   : 1.180   Min.   :21.00   Length:1000
##  1st Qu.: 9.10   1st Qu.: 4.555   1st Qu.:24.00   Class :character
##  Median :10.67   Median : 7.670   Median :29.00   Mode  :character
##  Mean   :10.68   Mean   : 9.105   Mean   :32.84
##  3rd Qu.:12.20   3rd Qu.:11.660   3rd Qu.:40.00
##  Max.   :22.37   Max.   :36.630   Max.   :81.00
```

```
unique(chatter.df$Resolved)
```

```
## [1] "No"  "Yes"
```

```
table(chatter.df$Resolved)
```

```
##
##  No Yes
## 339 661
```

Initial exploratory analysis reveals we are dealing with observations that have exclusively numeric data and one Y/N resolved classifier which is our parameter of interest.

```
colSums(is.na(chatter.df))
```

```
##      Worker     Max_msg     Min_msg   Exchanges  Total_time Time_length
##           0           0           0           0           0           0
##  Age_client    Resolved
##           0           0
```

All of our observations have recorded data for each variable which is ideal. We will change the Resolved category to a factor.

```
chatter.df$Resolved <- as.factor(chatter.df$Resolved)
```

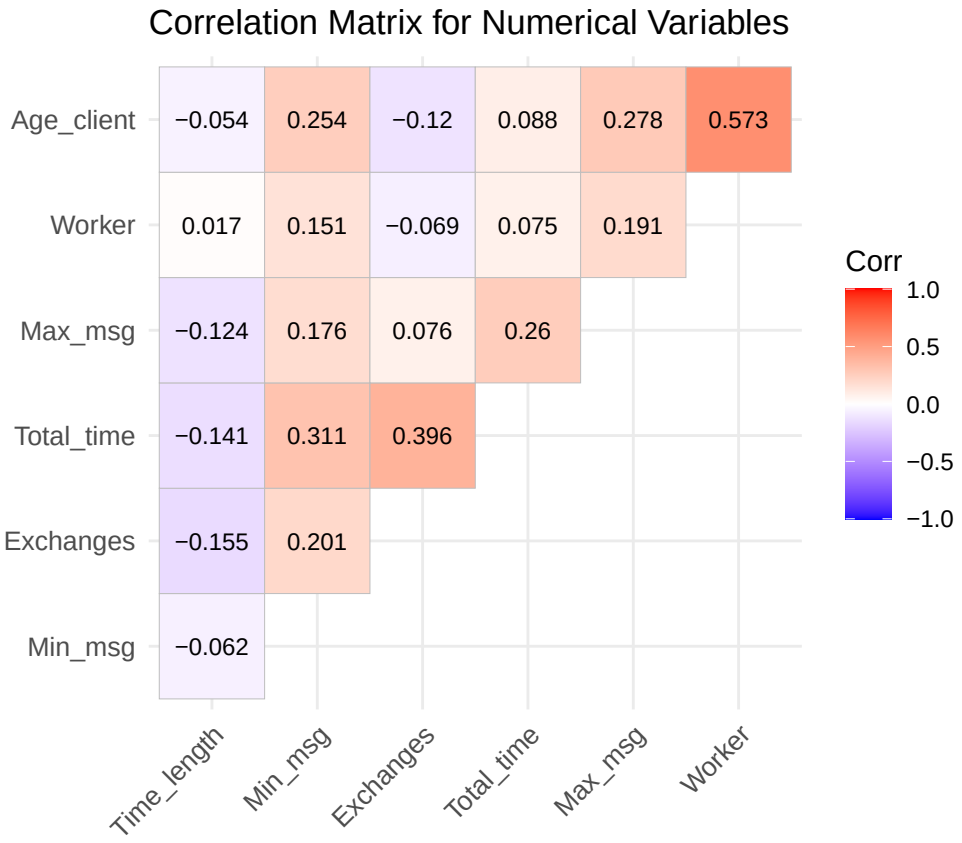
Let's make a correlation plot to try and better understand the context of our variables.

```
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 4.4.3
```

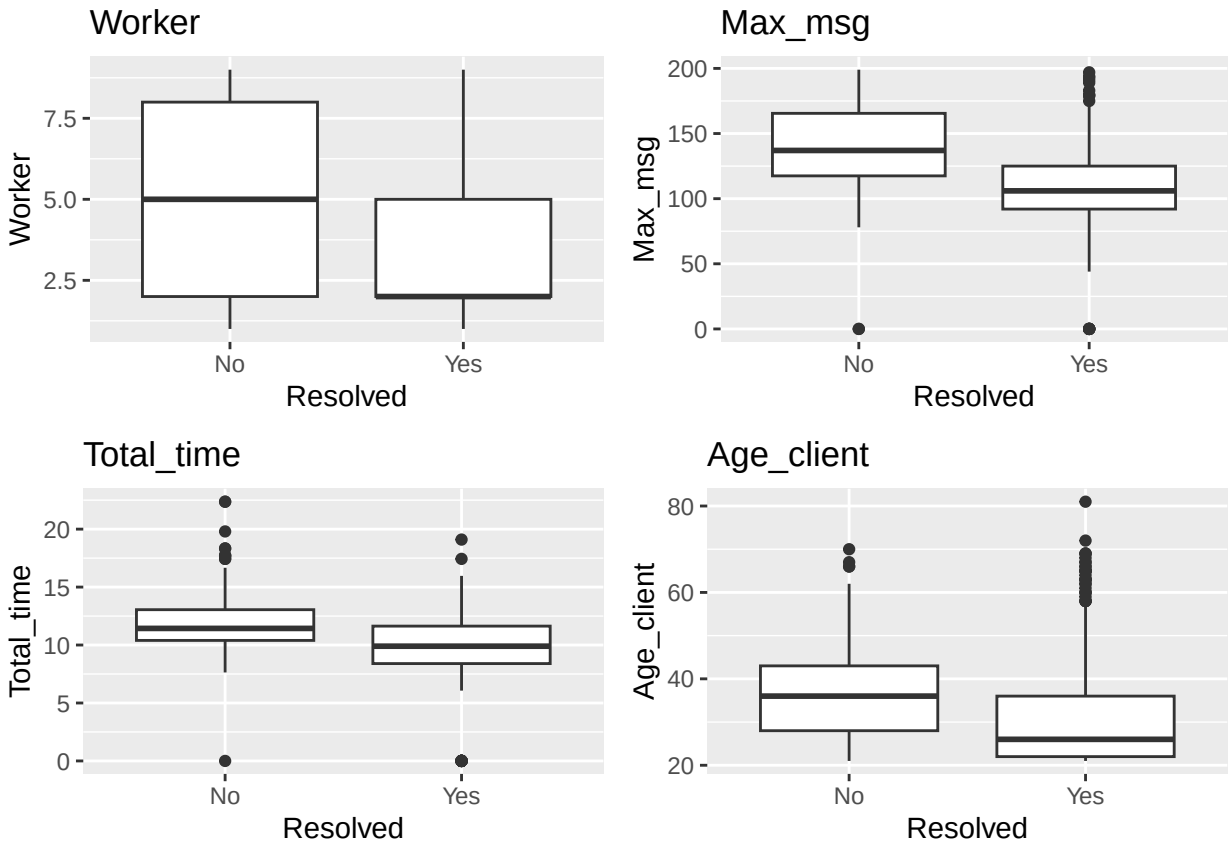
```
chatter_numeric <- subset(chatter.df, select = -Resolved)
corr_matrix <- cor(chatter_numeric)
ggcorrplot(corr_matrix,
           method = "square",
           lab = TRUE,
           lab_size = 3,        # Control label size
           lab_col = "black",   # Ensure contrast with tiles
```

```
            hc.order = TRUE,
            type = "upper",
            tl.cex = 10,
            digits = 3,
            title = "Correlation Matrix for Numerical Variables",
            show.legend = TRUE)
```



There are a few relationships of note however most variables are very weakly correlated. Older clients tend to receive higher employee self-assessment scores.

```
library(ggplot2)
library(gridExtra)
param_of_interest = list('Worker', "Max_msg", "Total_time", "Age_client")
i <- 1
plots <- list()
for (param in param_of_interest) {
  plots[[i]] <- ggplot(chatter.df, aes(x = Resolved, y = .data[[param]])) +
    geom_boxplot() +
    ggtitle(param)
  i <- i + 1
}

do.call(grid.arrange, c(plots, ncol = 2))
```



1.1 Creating a train/test split

Using an 80:20 train:test split, we are going to create a decision tree for the Chatter dataset.

```
set.seed(6969)
nrowdf <- nrow(chatter.df)
train<-sample(1:nrowdf, 0.8*nrowdf)
chatter.train <- chatter.df[train, ]
chatter.test <- chatter.df[-train, ]
tree.chatter <- tree(Resolved ~., data=chatter.train)
```

```
summary(tree.chatter)
```

```
##
## Classification tree:
## tree(formula = Resolved ~ ., data = chatter.train)
## Variables actually used in tree construction:
## [1] "Max_msg"    "Total_time" "Worker"     "Age_client" "Min_msg"
## [6] "Exchanges"
## Number of terminal nodes:  10
## Residual mean deviance:  0.8261 = 652.6 / 790
## Misclassification error rate: 0.1988 = 159 / 800
```

```
plot(tree.chatter)
text(tree.chatter, pretty = 1, cex = 0.7)
```

Our tree here has 14 terminal nodes and is highly complex, it is likely picking up a lot of noise from the training data and we should see it underperform when we fit our predictions.

```
tree.pred <- predict(tree.chatter, chatter.test, type="class")
tree.tab<-table(tree.pred, chatter.test$Resolved)
tree.tab
```

```
##
## tree.pred  No Yes
##       No   30   9
##       Yes  38 123
```

```
(tree.tab[1,1] + tree.tab[2,2])/sum(tree.tab)
```

```
## [1] 0.765
```

1.2 Tree Pruning.

We want to use cv.tree() function to decide the complexity level of our tree based on classification error.

```
RNGversion("3.5")
```

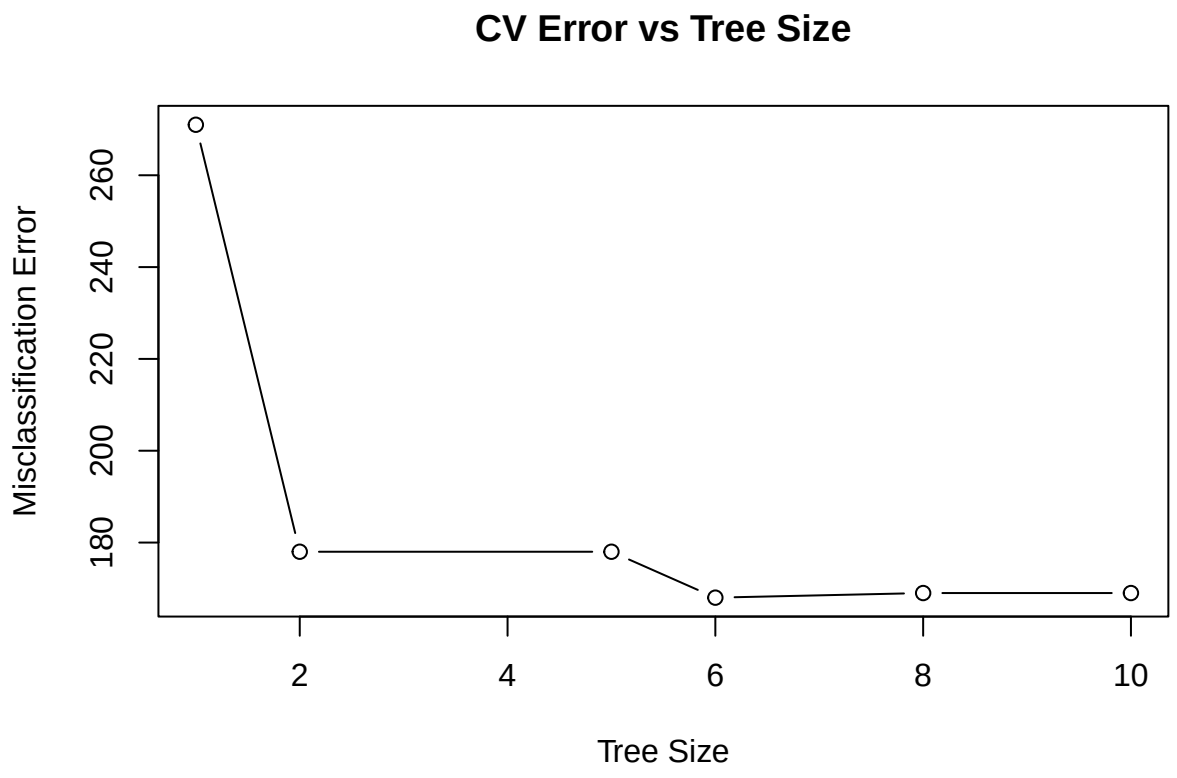
```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

```
set.seed(9)
cv.chatter <- cv.tree(tree.chatter, FUN=prune.misclass)
cv.chatter
```

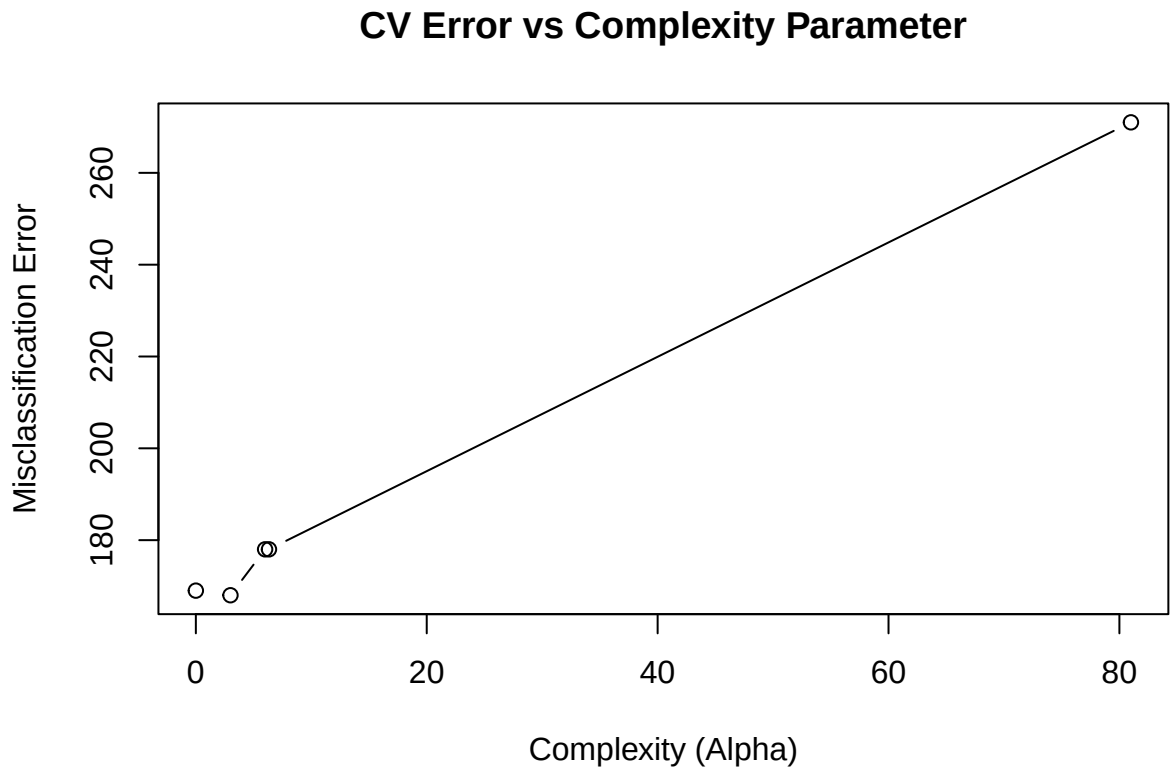
```
## $size
## [1] 10  8  6  5  2  1
##
## $dev
## [1] 169 169 168 178 178 271
##
## $k
## [1]      -Inf  0.000000  3.000000  6.000000  6.333333 81.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

From our overview, we see that the pruned tree with size 5 had the lowest test score (dev) in this case which is classification error of 173, however lets plot the CV error.

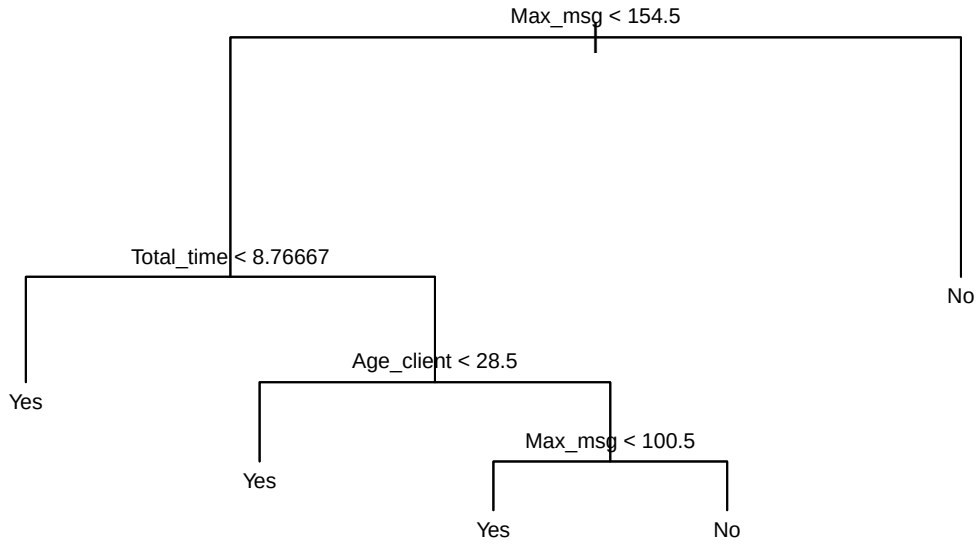
```
plot(cv.chatter$size,cv.chatter$dev, type="b", xlab="Tree Size", ylab="Misclassification Error",
     main="CV Error vs Tree Size")
```




```
plot(cv.chatter$k,cv.chatter$dev, type="b", xlab="Complexity (Alpha)", ylab="Misclassification Error",
     main="CV Error vs Complexity Parameter")
```




```
prune.chatter <- prune.misclass(tree.chatter, best=5)
plot(prune.chatter)
text(prune.chatter, pretty = 1, cex = 0.7)
```



Let's apply our predictions to our pruned tree and look at it's accuracy matrix.

```
prune.pred <- predict(prune.chatter, chatter.test, type="class")
prune.tab<-table(prune.pred, chatter.test$Resolved)
prune.tab
```

```
##
## prune.pred No Yes
##        No  53  35
##        Yes 15  97
```

```
(prune.tab[1,1] + prune.tab[2,2])/sum(prune.tab)
```

```
## [1] 0.75
```

1.3 Implementing a random forest.

When implementing a random forest, we tune the number of variables randomly selected from the total set of predictors to consider at each split in a decision tree.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.3
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

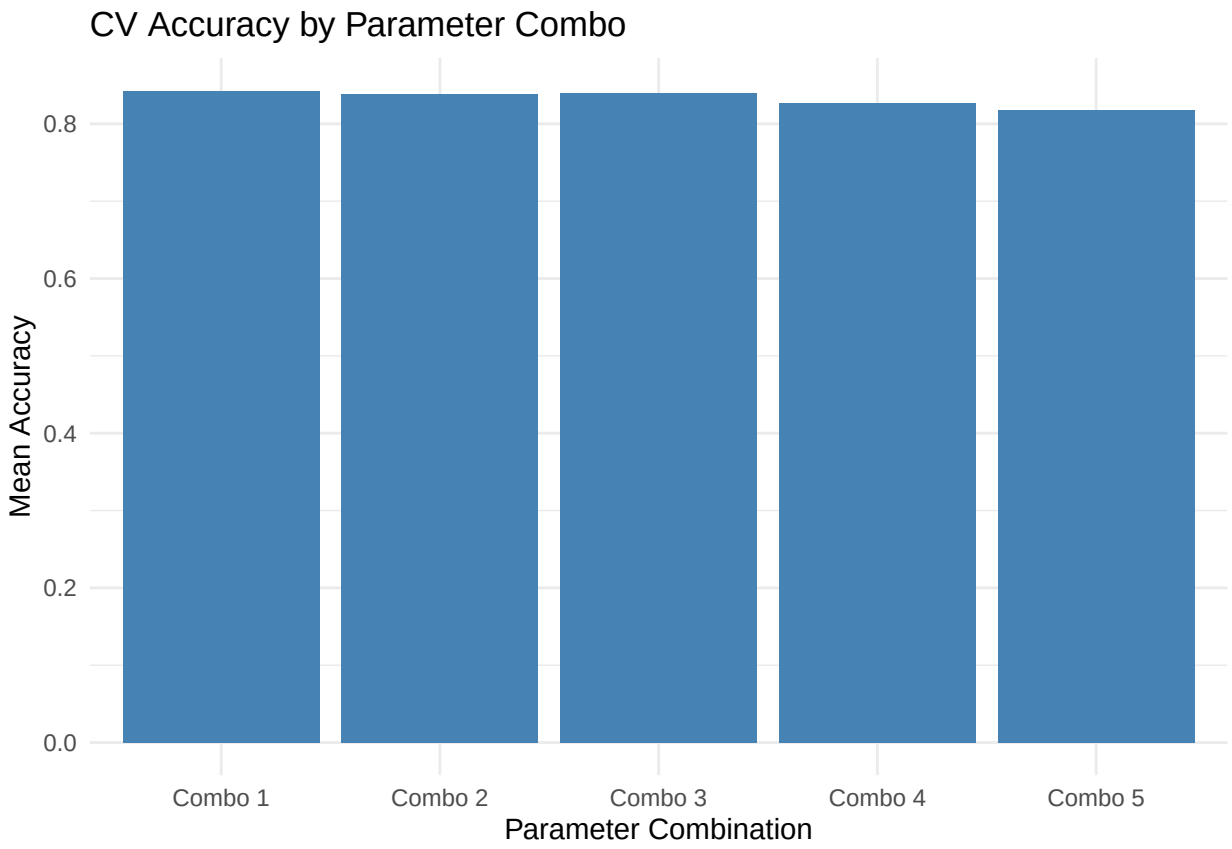
```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3

## Loading required package: lattice
```

```
set.seed(42)


folds <- createFolds(chatter.df$Resolved, k = 10, list = TRUE)


ntree <- 300
node_size <- c(3, 8, 10, 15, 20)

results <- data.frame()

# Loop over each parameter combo
for (param_i in 1:length(node_size)) {
  fold_accuracies <- c()

  for (fold_i in 1:10) {
    test_indices <- folds[[fold_i]]
    train_data <- chatter.df[-test_indices, ]
    test_data <- chatter.df[test_indices, ]

    # Train random forest with this param combo
    rf_model <- randomForest(Resolved ~ .,
                             data = train_data,
                             mtry = 3,
                             ntree = ntree,
                             nodesize = node_size[param_i])

    preds <- predict(rf_model, test_data)
    acc <- mean(preds == test_data$Resolved)
    fold_accuracies <- c(fold_accuracies, acc)
  }

  results <- rbind(results, data.frame(
    nodesize = node_size[param_i],
    ntree = ntree,
    mtry = 3,
    mean_accuracy = mean(fold_accuracies)
  ))
}
```

```
results$combo <- paste0("Combo ", 1:nrow(results))

ggplot(results, aes(x = combo, y = mean_accuracy)) +
  geom_col(fill = "steelblue") +
  labs(title = "CV Accuracy by Parameter Combo",
       x = "Parameter Combination",
       y = "Mean Accuracy") +
  theme_minimal()
```



I tested a variety of combinations of ntree and node_size and found that adjusting the parameters showed little variance in CV accuracy. Most of the forests ended up having a mean accuracy of approximately 80%. For this reason a random forest with less complexity will be used for our final forest. Combo 1 with nodesize 3 and ntrees 300.

```
final.forest <- randomForest(Resolved ~ .,
                             data = chatter.train,
                             mtry = 3,
                             ntree = ntree,
                             nodesize = 3

)

forest.pred <- predict(final.forest, chatter.test, type="class")
forest.tab <- table(forest.pred, chatter.test$Resolved)
forest.tab
```

```
##
## forest.pred  No Yes
```

```
##         No   51  12
##         Yes  17 120
```

```
(forest.tab[1,1] + forest.tab[2,2])/sum(forest.tab)
```

```
## [1] 0.855
```

1.3.1 Final evaluations.

After tuning nodesize using 10-fold cross-validation on the full chatter.df dataset and fixing mtry = 3, ntree = 300, the best-performing model used nodesize = 8. This model was then retrained on the full training set and evaluated on the hold-out test set. This model scored an 86% accuracy on the test set. This is a marginal improvement from our pruned tree which had 75% accuracy!

2 Diabetes dataset.

Researchers wanted to investigate whether patients show signs of diabetes according to certain criteria. Data for 768 women are stored in the file diabetesNA.csv. The variables are:

PREG: Number of times pregnant *PGC*: Plasma glucose concentration a 2 hours in an oral glucose tolerance test *DBP*: Diastolic blood pressure (mm Hg) *THICK*: Triceps skin fold thickness (mm) *INS*: 2-Hour serum insulin (µU/ml) *BMI*: Body mass index *PED*: Diabetes pedigree function *AGE*: Age (years) *DIAB*: Binary response (0: no diabetes; 1: diabetes)

2.1 Exploratory analysis.

```
db.df <- read.csv("diabetesNA.csv", header = T)
colSums(is.na(db.df))
```

```
##  PREG   PGC   DBP THICK   INS   BMI   PED   AGE  DIAB
##     0     5    35   227   374    11     0     0     0
```

```
summary(db.df)
```

```
##       PREG             PGC             DBP             THICK
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:22.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :29.00
##  Mean   : 3.845   Mean   :121.7   Mean   : 72.41   Mean   :29.15
##  3rd Qu.: 6.000   3rd Qu.:141.0   3rd Qu.: 80.00   3rd Qu.:36.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##                   NA's   :5       NA's   :35       NA's   :227
##       INS              BMI             PED              AGE
##  Min.   : 14.00   Min.   :18.20   Min.   :0.0780   Min.   :21.00
##  1st Qu.: 76.25   1st Qu.:27.50   1st Qu.:0.2437   1st Qu.:24.00
##  Median :125.00   Median :32.30   Median :0.3725   Median :29.00
##  Mean   :155.55   Mean   :32.46   Mean   :0.4719   Mean   :33.24
##  3rd Qu.:190.00   3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200   Max.   :81.00
##  NA's   :374      NA's   :11
##       DIAB
```

```
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
##
```

All of our variables are numeric besides our binary classifier variable DIAB. Since SVM's are distance based classification tools we need to ensure all quantitiative data is scaled to a Z-score.First we should the scale the data and also revert the binary classifier into a factor with 2 levels.

```
predictors_standardised <- scale(db.df[, -9])
standard.df <- data.frame(predictors_standardised, DIAB = as.factor(db.df$DIAB))
```

```
library(ggplot2)
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.4.3
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
# Pairwise scatterplots + density
ggpairs(standard.df[, c("PGC", "BMI", "AGE", "DIAB")],
        aes(color = DIAB, alpha = 0.5)) +
  theme_bw()
```

```
## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_density()`).
```

```
## Warning in ggally_statistic(data = data, mapping = mapping, na.rm = na.rm, :
## Removed 16 rows containing missing values
```

```
## Warning in ggally_statistic(data = data, mapping = mapping, na.rm = na.rm, :
## Removed 5 rows containing missing values
```

```
## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

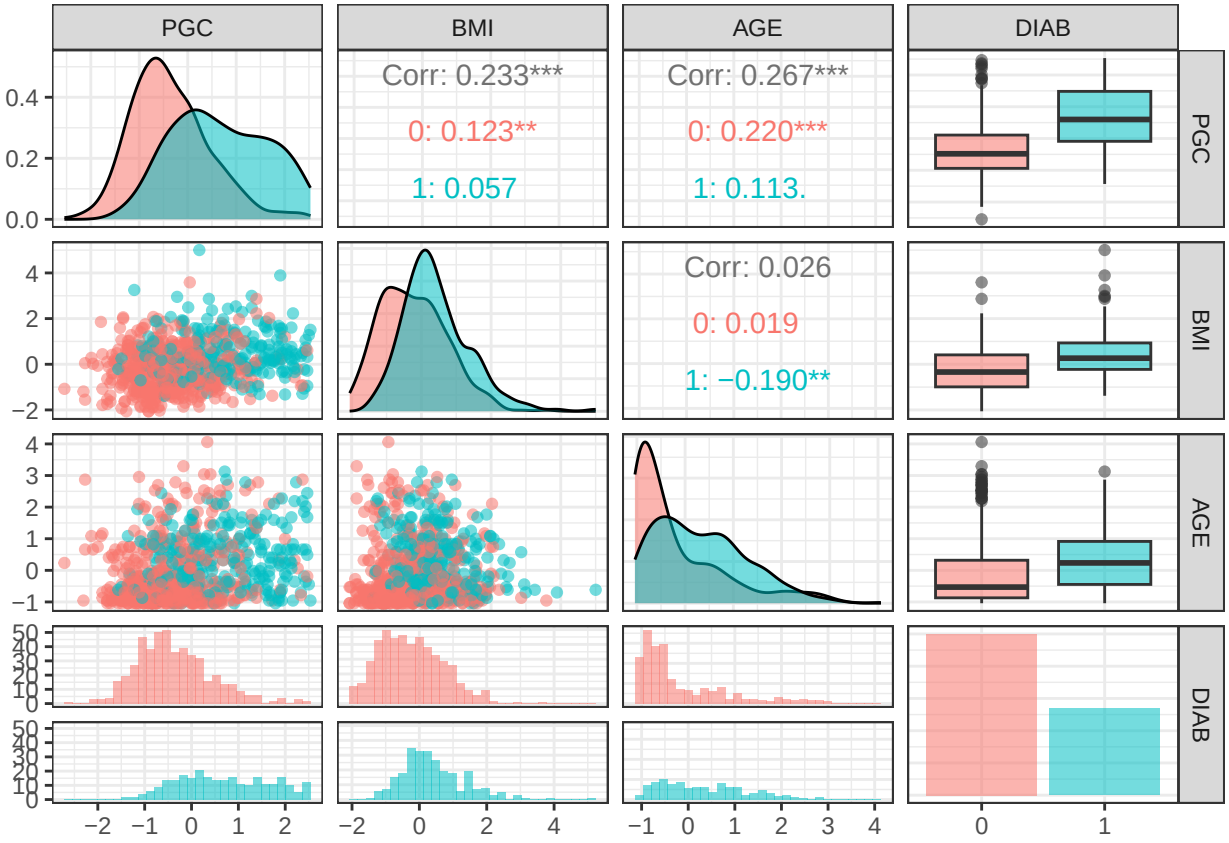
```
## Warning: Removed 16 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

```
## Warning: Removed 11 rows containing non-finite outside the scale range
## (`stat_density()`).
```

```
## Warning in ggally_statistic(data = data, mapping = mapping, na.rm = na.rm, :
## Removed 11 rows containing missing values
```

```
## Warning: Removed 11 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

```
## Warning: Removed 5 rows containing missing values or values outside the scale range
## (`geom_point()`).

## Warning: Removed 11 rows containing missing values or values outside the scale range
## (`geom_point()`).

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 5 rows containing non-finite outside the scale range
## (`stat_bin()`).

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 11 rows containing non-finite outside the scale range
## (`stat_bin()`).

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



These density plots tell us critical information about the relationship of our predictors as to how they pertain to a diabetic diagnosis. The density plots are overlapping and not unimodal indicating there are likely (multidimensional) polynomial relationships. The kernel function type which we select to find our ideal boundary lines for SVMs affects the way our model separates the data points. Based on this knowledge we will likely be fitting a polynomial or radial kernel function.

2.2 Creating a train/test split.

We are going to use a 60/40 split to train and validate our SVM.

```
set.seed(69)
nrowdf <- nrow(standard.df)
trainsplit <- sample(1:nrowdf, 0.6*nrowdf)
train_data <- standard.df[trainsplit, ]
test_data <- standard.df[-trainsplit, ]

comp_test <- test_data[complete.cases(test_data), ]
```

2.3 Fitting an initial SVM.

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.4.3
```

```
svm_poly <- tune(
  svm,
  DIAB ~ .,
  data = train_data,
  kernel = "polynomial",
  na.action = na.omit,
  ranges = list(
    cost = c(0.1, 1, 10, 100),
    gamma = c(0.01, 0.1, 1)
  )
)
```

```
svm_radial <- tune(
  svm,
  DIAB ~ .,
  data = train_data,
  kernel = "radial",
  na.action = na.omit,
  ranges = list(
    cost = c(0.1, 1, 10, 100),
    gamma = c(0.01, 0.1, 1)
  )
)
```

```
svm_radial$best.performance
```

```
## [1] 0.1839097
```

```
svm_poly$best.performance
```

```
## [1] 0.2286147
```

It turns out our best performing SVM used a radial kernel leading to least error after cross validation.

```
pred_scores <- predict(
  svm_radial$best.model,
  newdata = comp_test,
  decision.values = TRUE
)
decision_values <- attributes(pred_scores)$decision.values
```

2.4 Calculating a confusion matrix.

```
pred = predict(svm_radial$best.model, newdata = comp_test)
conf_matrix <- table(
  true = comp_test$DIAB,
  pred = pred
)

conf_matrix
```

```
##     pred
## true  0  1
##    0 96 21
##    1 23 23
```

```
(conf_matrix[1,1] + conf_matrix[2,2]) / sum(conf_matrix)
```

```
## [1] 0.7300613
```

Our final SVM had a true prediction rate of 73% on the test set. We clearly notice an imbalance in class detection percentages. While the model correctly identifies 96 non-diabetic cases, it misses 50% of diabetic cases and incorrectly flags 21 non-diabetic individuals as diabetic. These factors compromise its clinical utility for diabetes diagnosis.