Statistical Learning Final Project

Alison Lanski, Ken Nagle, Marisa Roman, Russ Thomas

10/31/2018

Load the Data Set and Perform EDA

Load the `spambase` data set:

```
# Load the spambase data set into a data frame
spambase = read.csv("~/GitHub/Stat-Learning-2018-Fall/Classification_Final_Data/spambase/spambas
e.data",
                    header = FALSE)

# Glimpse the data set
glimpse(spambase)
```

```
## Observations: 4,601
## Variables: 58
## $ V1  <dbl> 0.00, 0.21, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.06...
## $ V2  <dbl> 0.64, 0.28, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.12...
## $ V3  <dbl> 0.64, 0.50, 0.71, 0.00, 0.00, 0.00, 0.00, 0.00, 0.46, 0.77...
## $ V4  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V5  <dbl> 0.32, 0.14, 1.23, 0.63, 0.63, 1.85, 1.92, 1.88, 0.61, 0.19...
## $ V6  <dbl> 0.00, 0.28, 0.19, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.32...
## $ V7  <dbl> 0.00, 0.21, 0.19, 0.31, 0.31, 0.00, 0.00, 0.00, 0.30, 0.38...
## $ V8  <dbl> 0.00, 0.07, 0.12, 0.63, 0.63, 1.85, 0.00, 1.88, 0.00, 0.00...
## $ V9  <dbl> 0.00, 0.00, 0.64, 0.31, 0.31, 0.00, 0.00, 0.00, 0.92, 0.06...
## $ V10 <dbl> 0.00, 0.94, 0.25, 0.63, 0.63, 0.00, 0.64, 0.00, 0.76, 0.00...
## $ V11 <dbl> 0.00, 0.21, 0.38, 0.31, 0.31, 0.00, 0.96, 0.00, 0.76, 0.00...
## $ V12 <dbl> 0.64, 0.79, 0.45, 0.31, 0.31, 0.00, 1.28, 0.00, 0.92, 0.64...
## $ V13 <dbl> 0.00, 0.65, 0.12, 0.31, 0.31, 0.00, 0.00, 0.00, 0.00, 0.25...
## $ V14 <dbl> 0.00, 0.21, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V15 <dbl> 0.00, 0.14, 1.75, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.12...
## $ V16 <dbl> 0.32, 0.14, 0.06, 0.31, 0.31, 0.00, 0.96, 0.00, 0.00, 0.00...
## $ V17 <dbl> 0.00, 0.07, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V18 <dbl> 1.29, 0.28, 1.03, 0.00, 0.00, 0.00, 0.32, 0.00, 0.15, 0.12...
## $ V19 <dbl> 1.93, 3.47, 1.36, 3.18, 3.18, 0.00, 3.85, 0.00, 1.23, 1.67...
## $ V20 <dbl> 0.00, 0.00, 0.32, 0.00, 0.00, 0.00, 0.00, 0.00, 3.53, 0.06...
## $ V21 <dbl> 0.96, 1.59, 0.51, 0.31, 0.31, 0.00, 0.64, 0.00, 2.00, 0.71...
## $ V22 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V23 <dbl> 0.00, 0.43, 1.16, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.19...
## $ V24 <dbl> 0.00, 0.43, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.00...
## $ V25 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V26 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V27 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V28 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V29 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V30 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V31 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V32 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V33 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.00...
## $ V34 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V35 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V36 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V37 <dbl> 0.00, 0.07, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V38 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V39 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V40 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V41 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V42 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V43 <dbl> 0.00, 0.00, 0.12, 0.00, 0.00, 0.00, 0.00, 0.00, 0.30, 0.00...
## $ V44 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.06...
## $ V45 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V46 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V47 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V48 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V49 <dbl> 0.000, 0.000, 0.010, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
## $ V50 <dbl> 0.000, 0.132, 0.143, 0.137, 0.135, 0.223, 0.054, 0.206, 0....
## $ V51 <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
```

```
## $ V52 <dbl> 0.778, 0.372, 0.276, 0.137, 0.135, 0.000, 0.164, 0.000, 0....
## $ V53 <dbl> 0.000, 0.180, 0.184, 0.000, 0.000, 0.000, 0.054, 0.000, 0....
## $ V54 <dbl> 0.000, 0.048, 0.010, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
## $ V55 <dbl> 3.756, 5.114, 9.821, 3.537, 3.537, 3.000, 1.671, 2.450, 9....
## $ V56 <int> 61, 101, 485, 40, 40, 15, 4, 11, 445, 43, 6, 11, 61, 7, 24...
## $ V57 <int> 278, 1028, 2259, 191, 191, 54, 112, 49, 1257, 749, 21, 184...
## $ V58 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
```

Assign meaningful column names:

```r
# Columns are assumed to be ordered as stated in the spambase.names file
new_col_names = c("word_freq_make",
                  "word_freq_address",
                  "word_freq_all",
                  "word_freq_3d",
                  "word_freq_our",
                  "word_freq_over",
                  "word_freq_remove",
                  "word_freq_internet",
                  "word_freq_order",
                  "word_freq_mail",
                  "word_freq_receive",
                  "word_freq_will",
                  "word_freq_people",
                  "word_freq_report",
                  "word_freq_addresses",
                  "word_freq_free",
                  "word_freq_business",
                  "word_freq_email",
                  "word_freq_you",
                  "word_freq_credit",
                  "word_freq_your",
                  "word_freq_font",
                  "word_freq_000",
                  "word_freq_money",
                  "word_freq_hp",
                  "word_freq_hpl",
                  "word_freq_george",
                  "word_freq_650",
                  "word_freq_lab",
                  "word_freq_labs",
                  "word_freq_telnet",
                  "word_freq_857",
                  "word_freq_data",
                  "word_freq_415",
                  "word_freq_85",
                  "word_freq_technology",
                  "word_freq_1999",
                  "word_freq_parts",
                  "word_freq_pm",
                  "word_freq_direct",
                  "word_freq_cs",
                  "word_freq_meeting",
                  "word_freq_original",
                  "word_freq_project",
                  "word_freq_re",
                  "word_freq_edu",
                  "word_freq_table",
                  "word_freq_conference",
                  "char_freq_semicolon",
                  "char_freq_open_paren",
                  "char_freq_open_square",
                  "char_freq_exclamation",
```

```
                  "char_freq_dollar",
                  "char_freq_pound",
                  "capital_run_length_average",
                  "capital_run_length_longest",
                  "capital_run_length_total",
                  "spam")

colnames(spambase) = new_col_names
```

```
# Convert spam column to a factor (plays better with randomForest function)
spambase <- spambase %>%
  mutate(spam = as.factor(ifelse(spam == 1,
                                 "Yes",
                                 "No")))
```

Review distribution of observations:

```
table(spambase$spam)
```

```
##
##   No  Yes
## 2788 1813
```

Dictionary for reference:

> SPAM E-MAIL DATABASE ATTRIBUTES (in .names format)
>
> 48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
>
> 6 continuous real [0,100] attributes of type char_freq_CHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
>
> 1 continuous real [1,…] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
>
> 1 continuous integer [1,…] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
>
> 1 continuous integer [1,…] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
>
> 1 nominal {0,1} class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.
>
> For more information, see file 'spambase.DOCUMENTATION' at the UCI Machine Learning Repository: http://www.ics.uci.edu/~mlearn/MLRepository.html (http://www.ics.uci.edu/~mlearn/MLRepository.html)

Assess Missingness

According to the documentation, there are no missing variable values. Confirm:

```
sum(is.na(spambase))
```

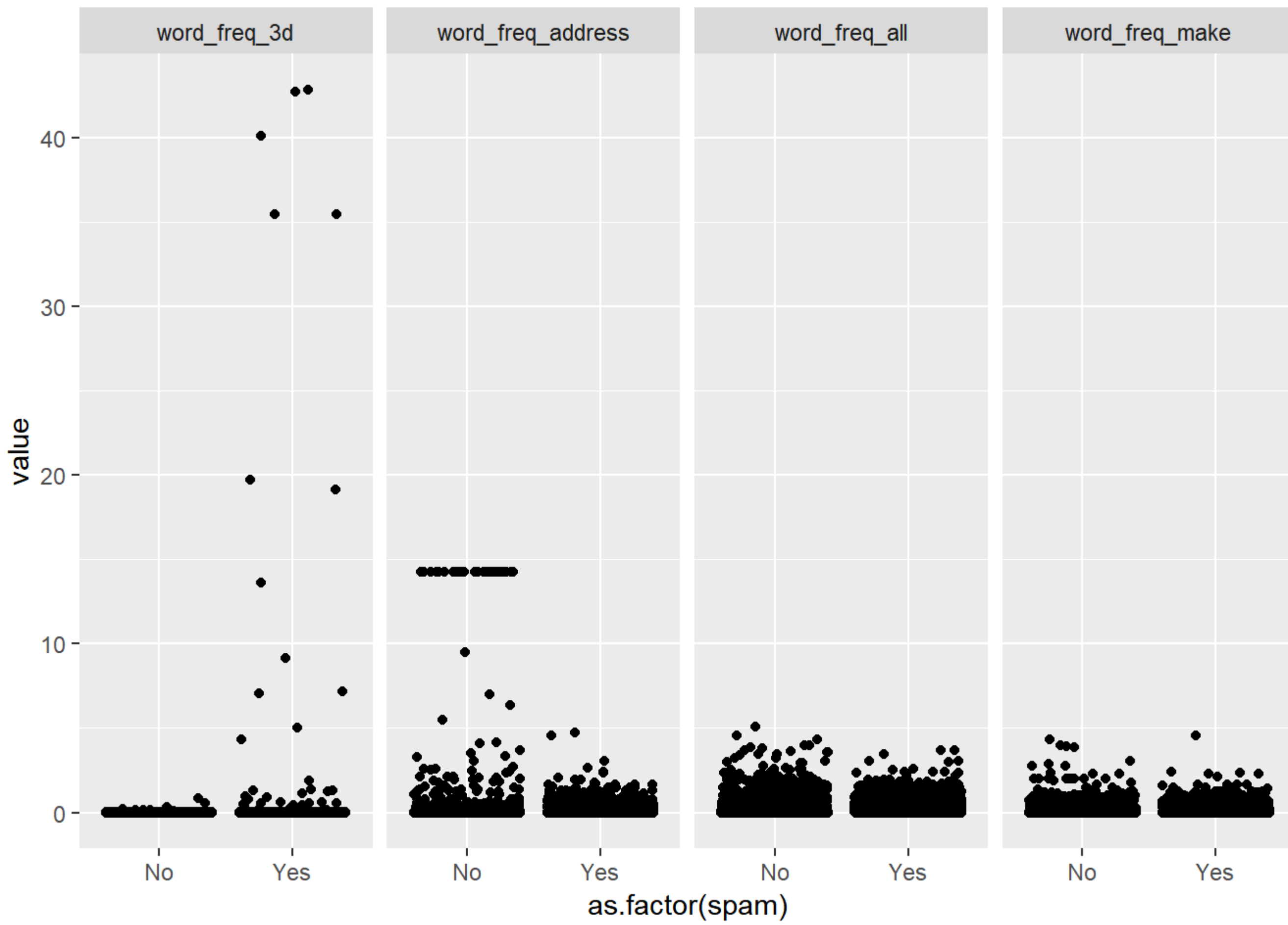
```
## [1] 0
```

Review Predictor/Response Relationships

Generate jitter plots of each potential predictor, grouped by whether the email is spam or not.

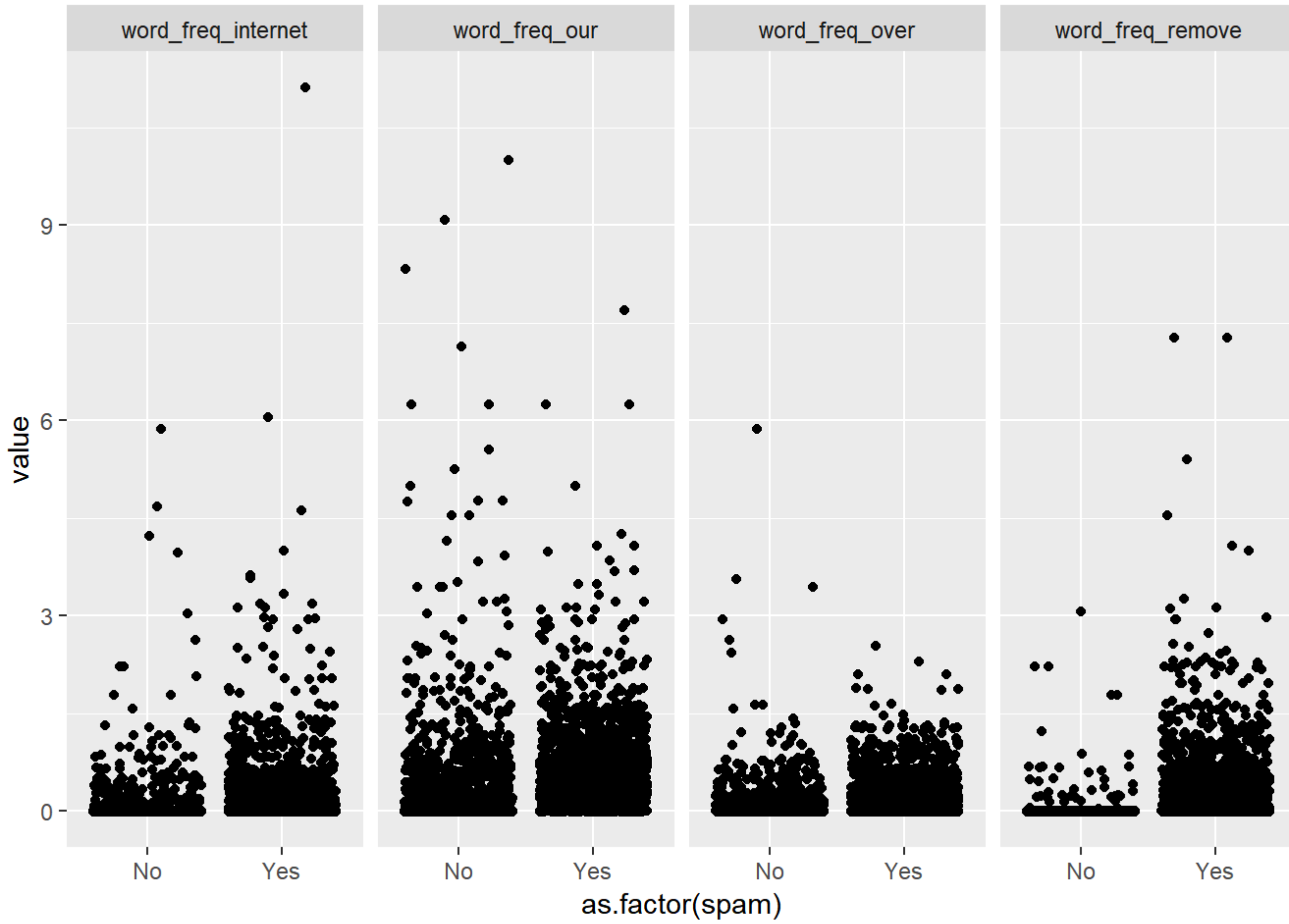
Columns 1:4

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(1:4, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



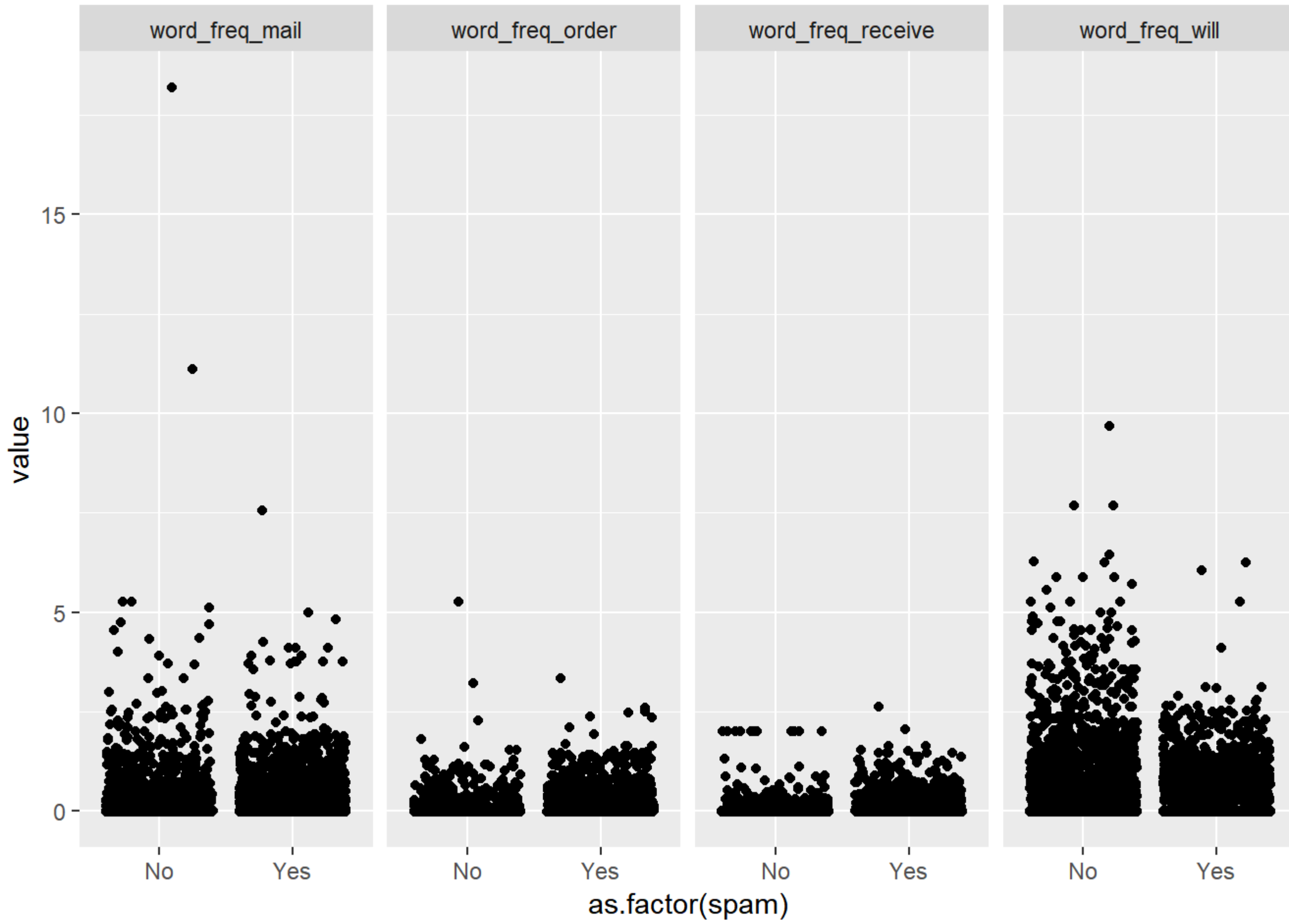
Columns 5:8

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(5:8, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



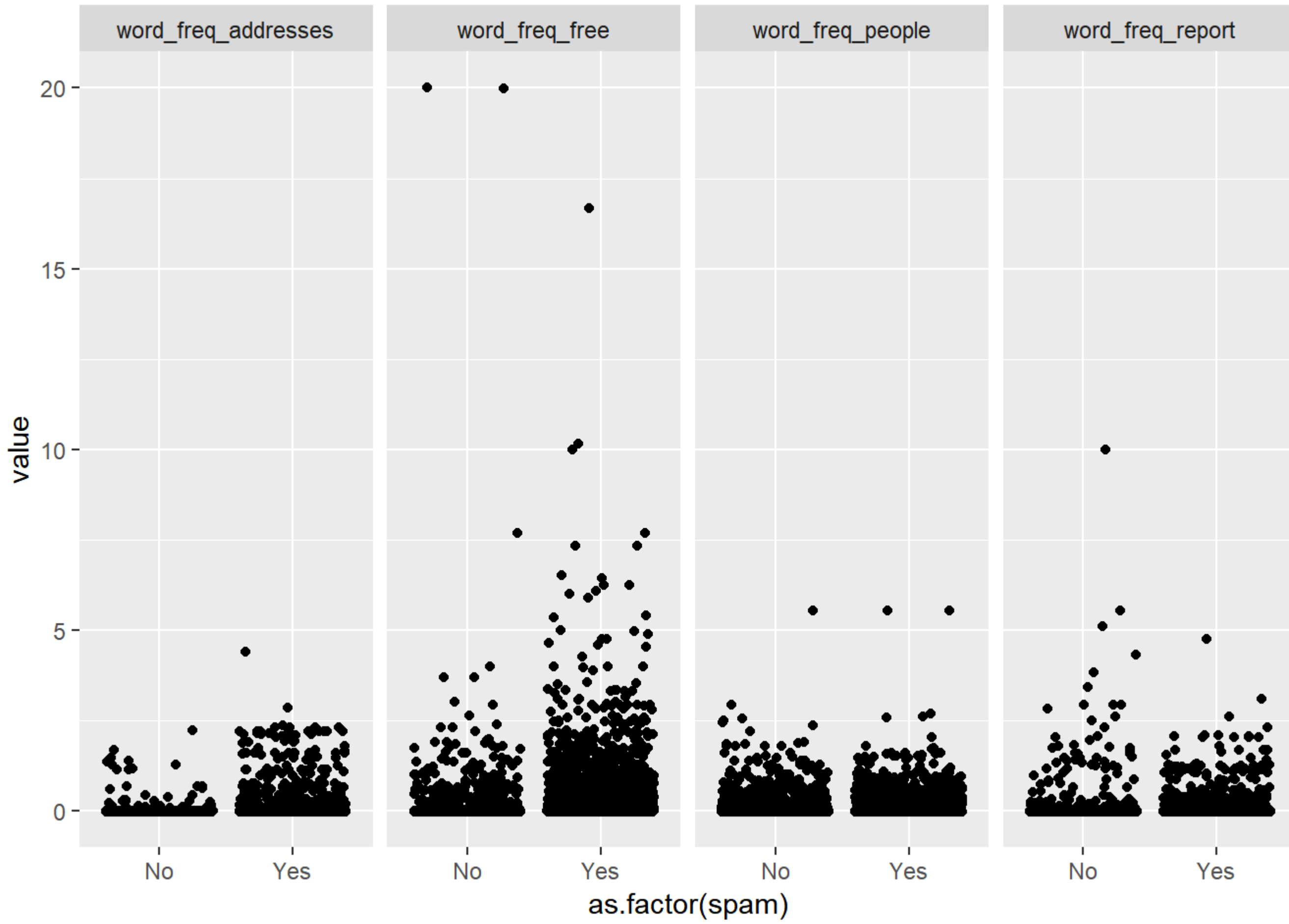
Columns 9:12

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(9:12, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plote of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



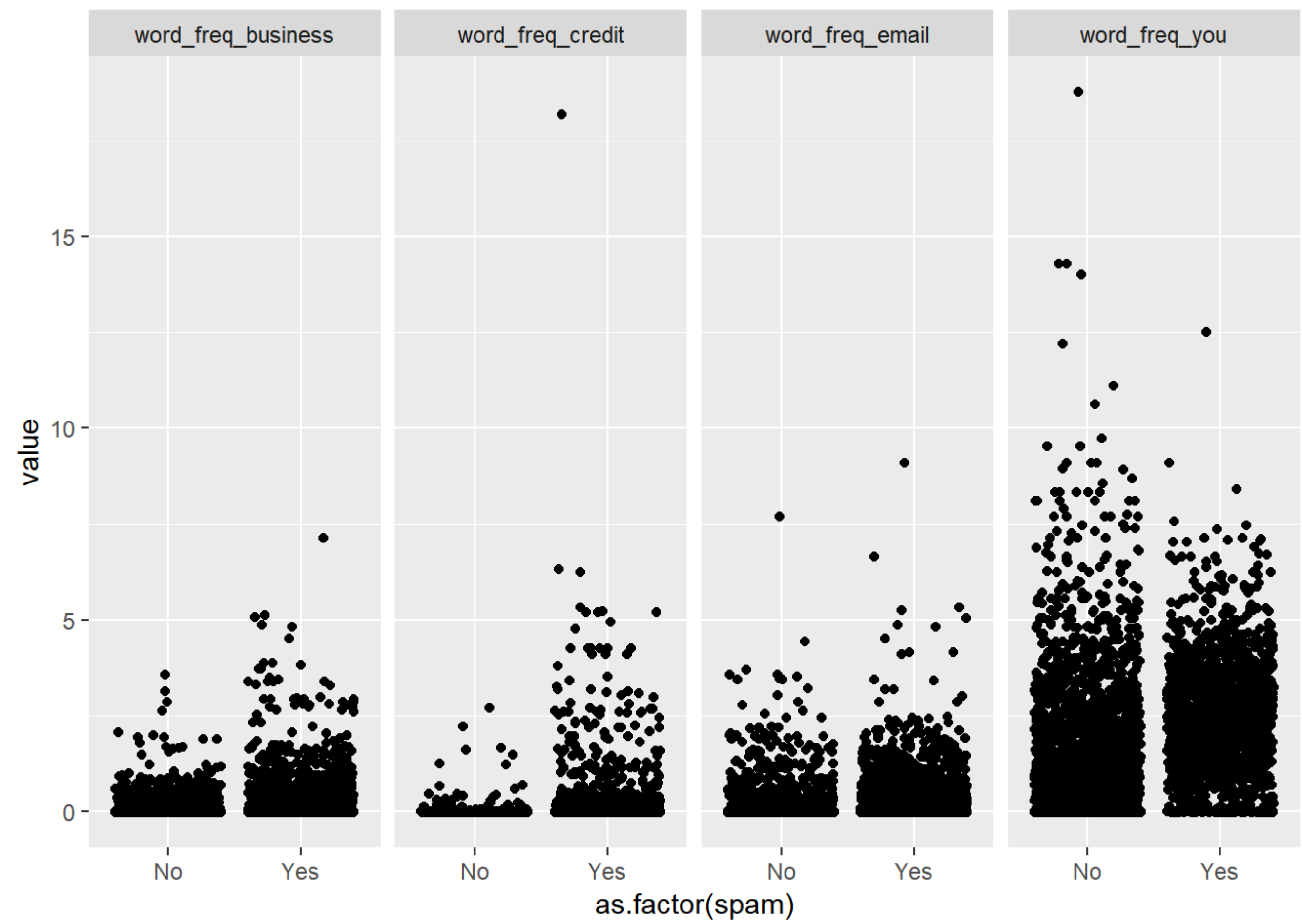
Columns 13:16

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(13:16, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



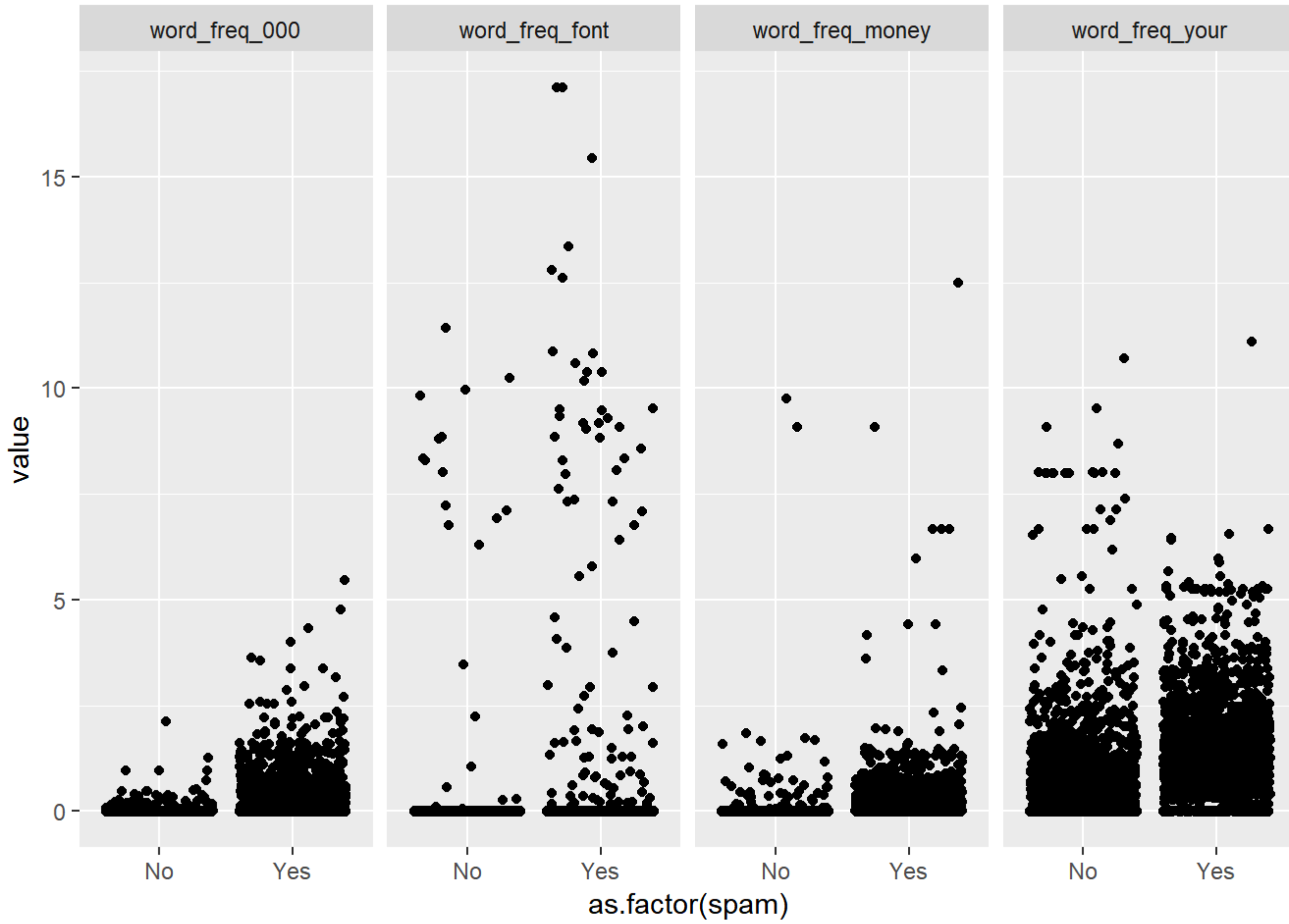
Columns 17:20

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(17:20, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



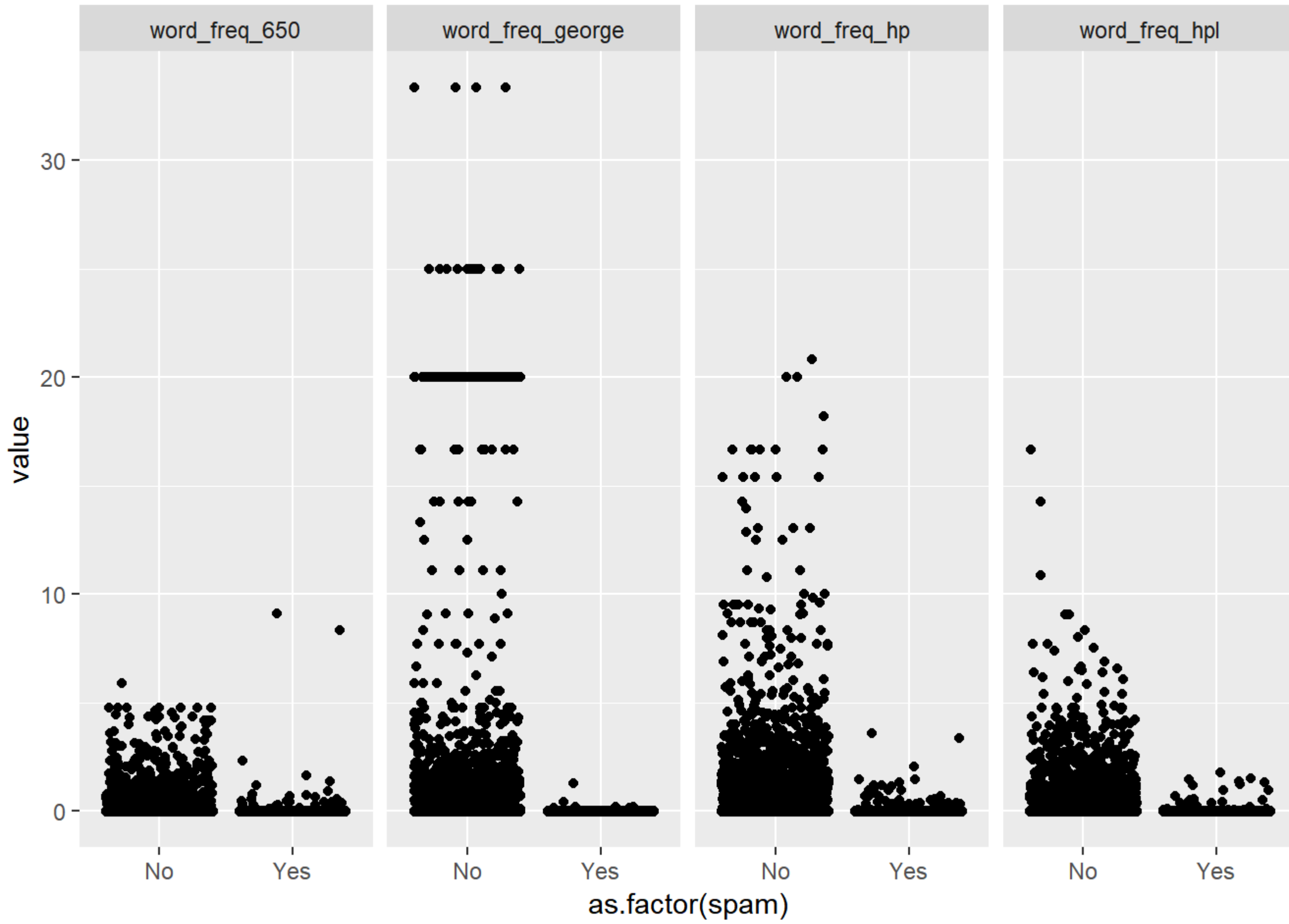
Columns 21:24

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(21:24, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



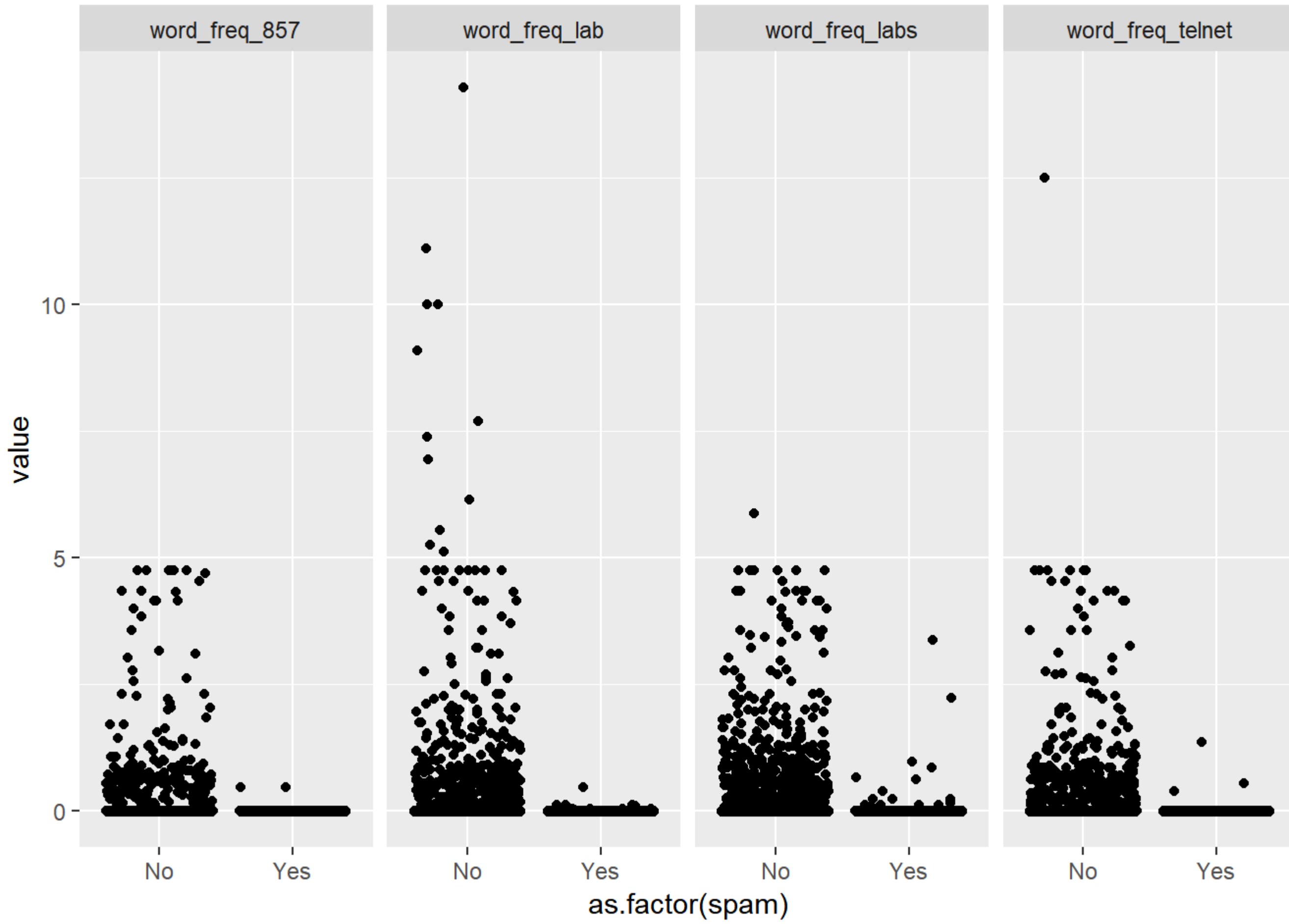
Columns 25:28

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(25:28, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



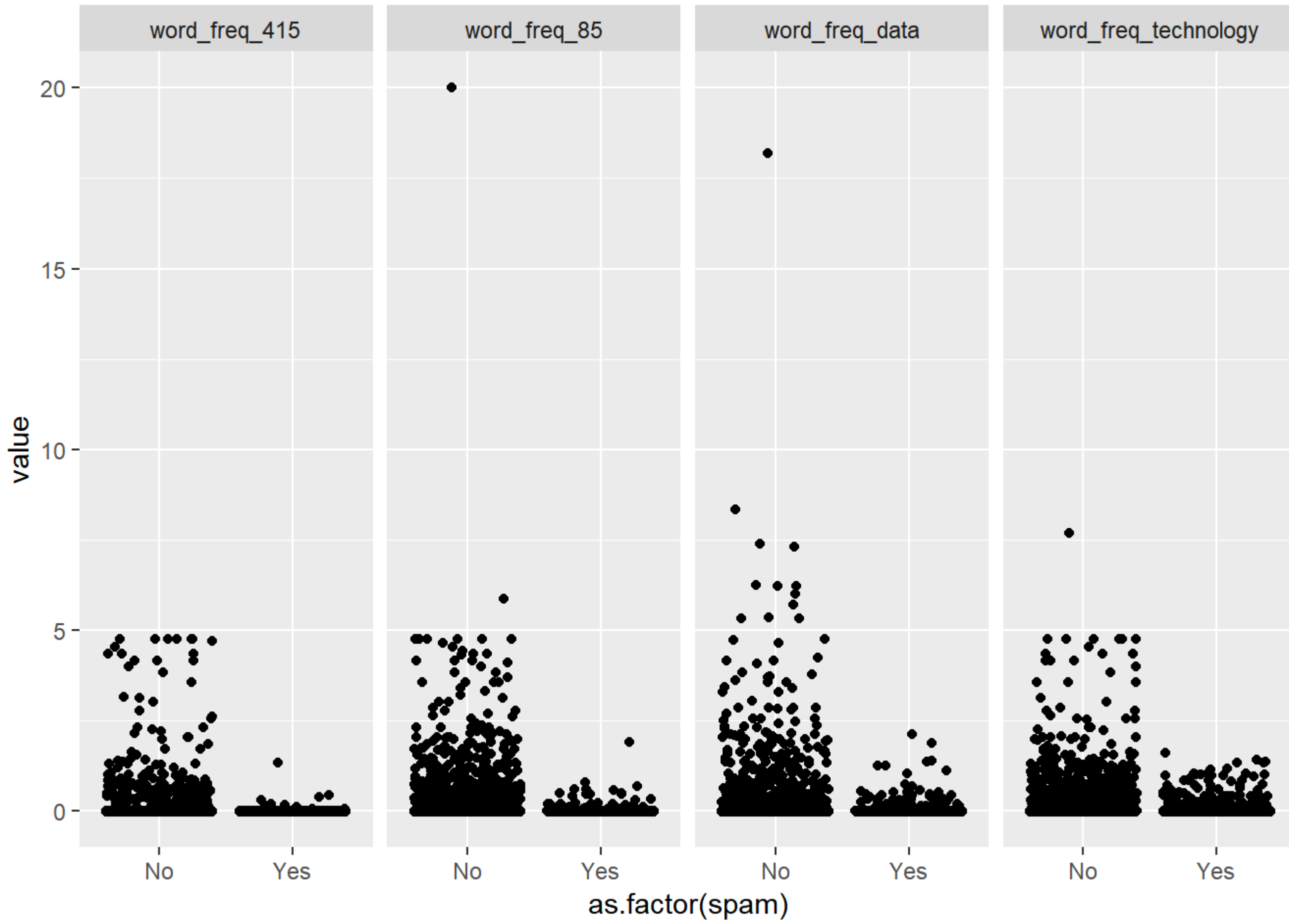
Columns 29:32

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(29:32, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



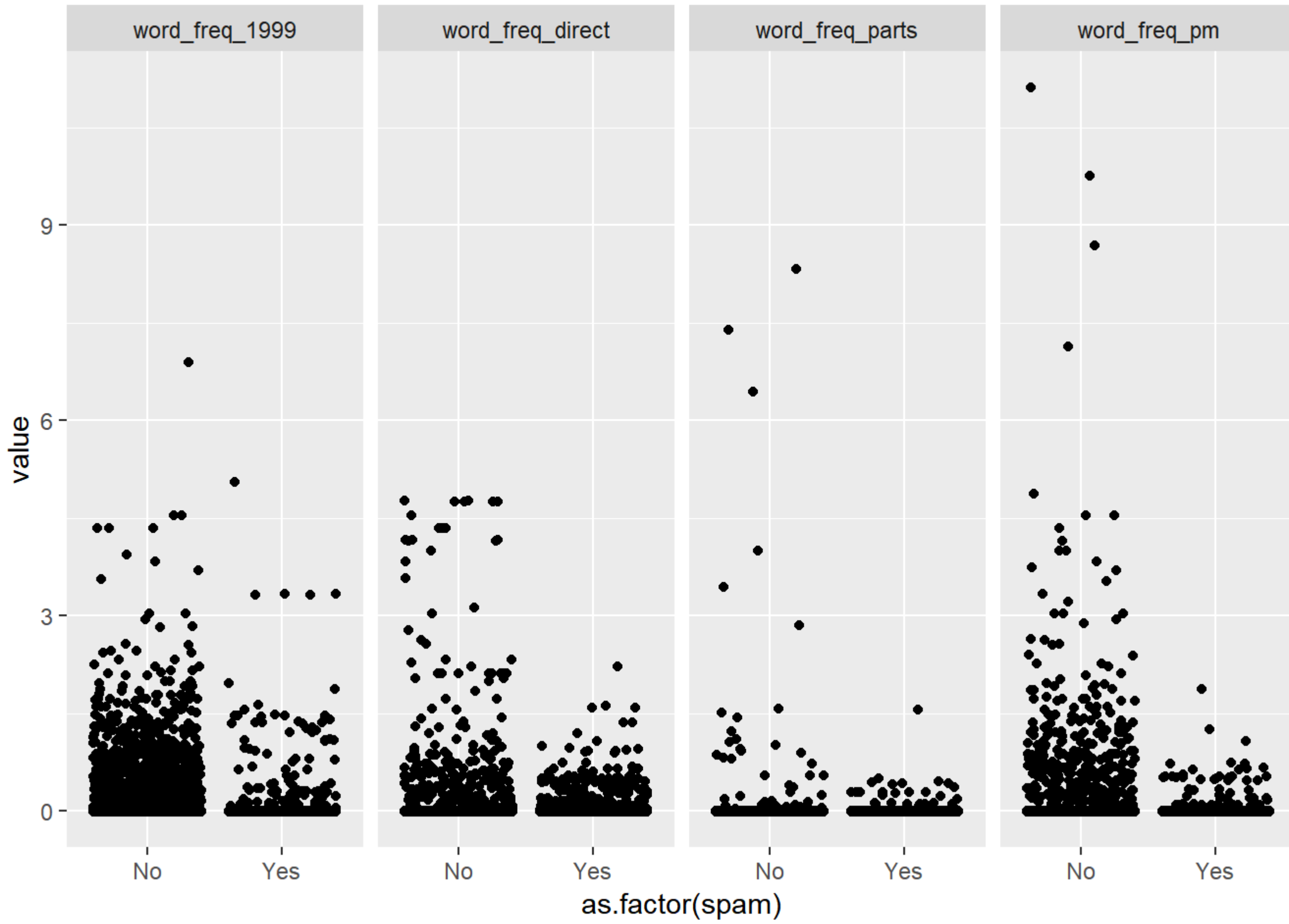
Columns 33:36

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(33:36, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



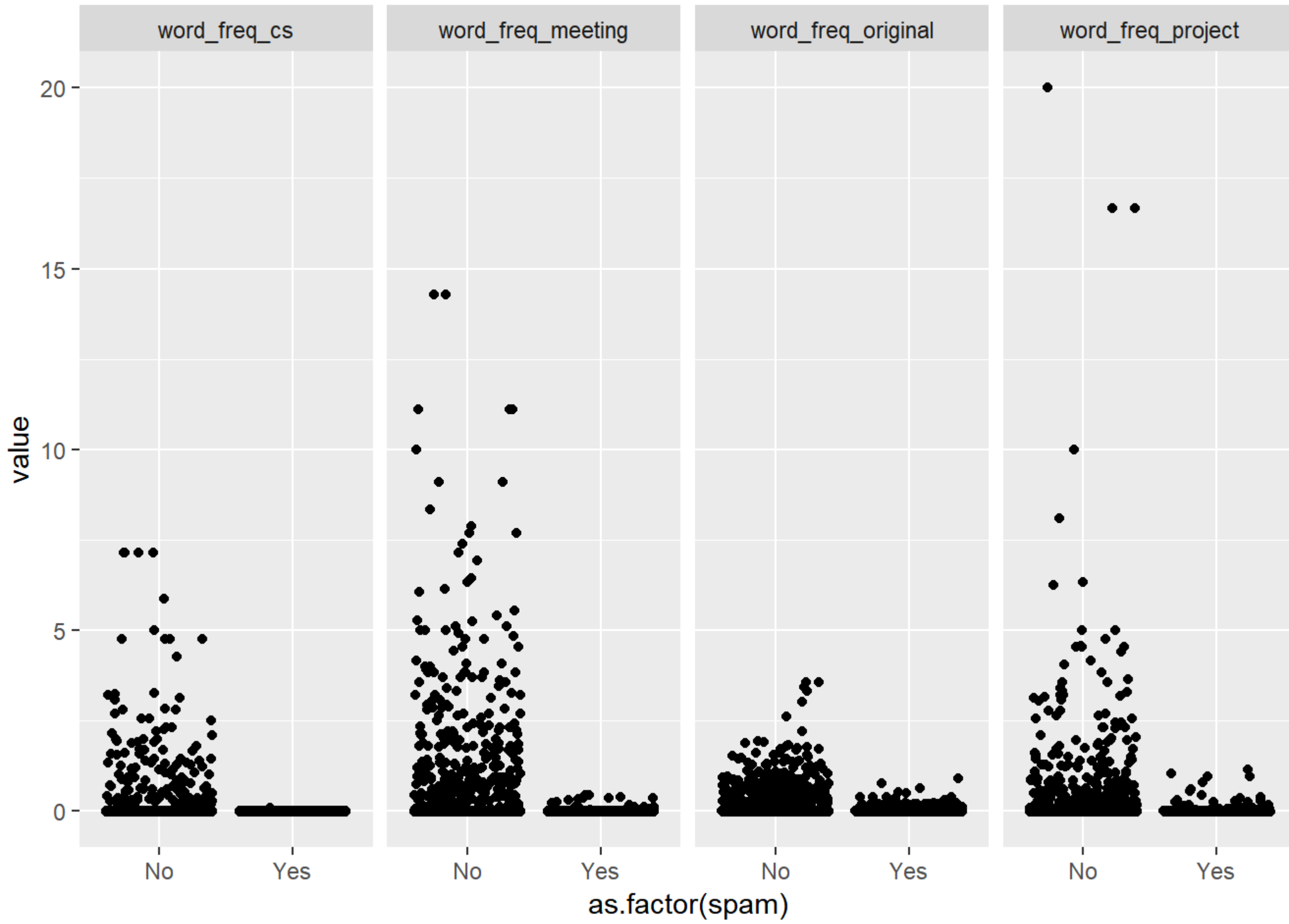
Columns 37:40

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(37:40, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



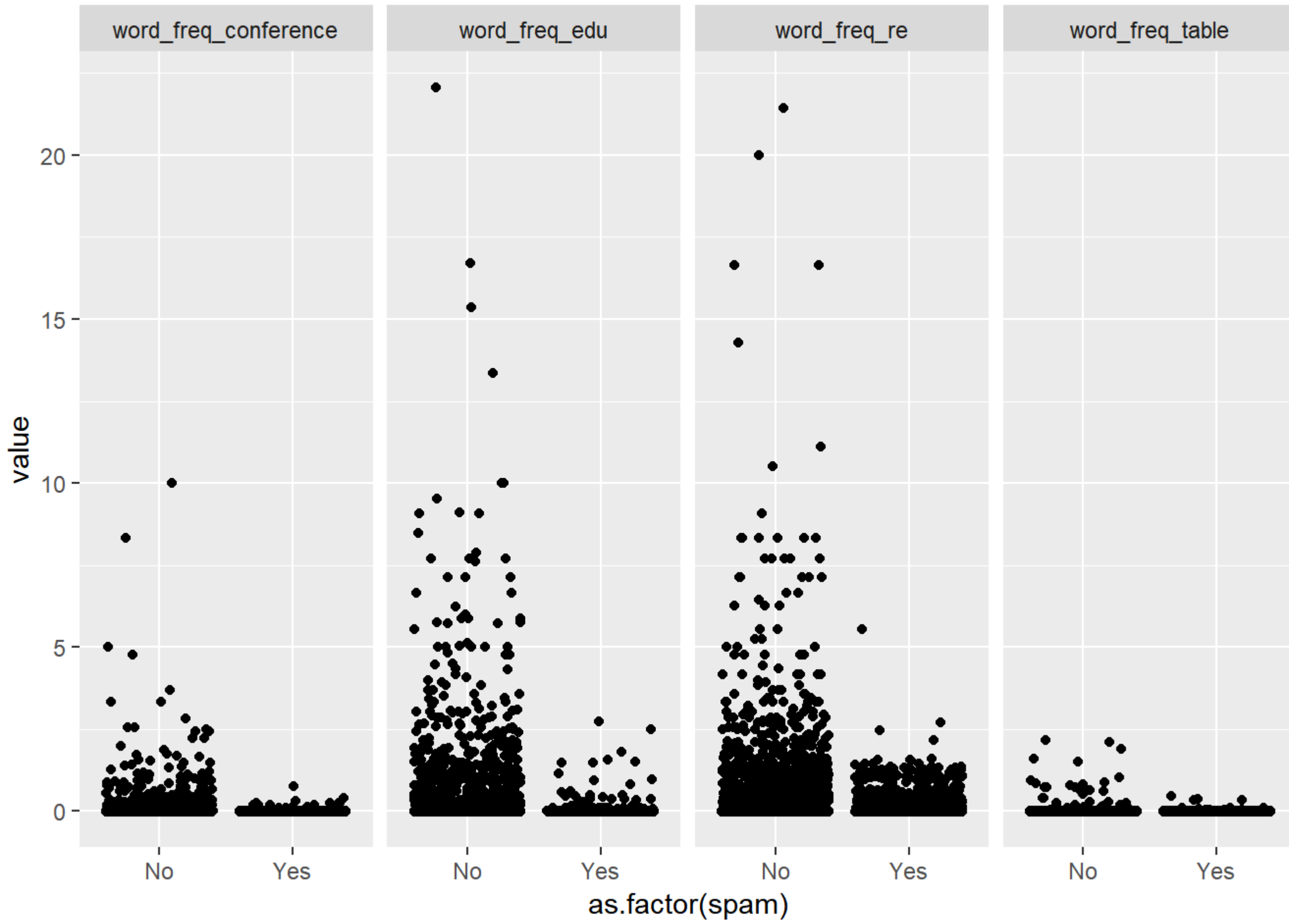
Columns 41:44

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(41:44, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



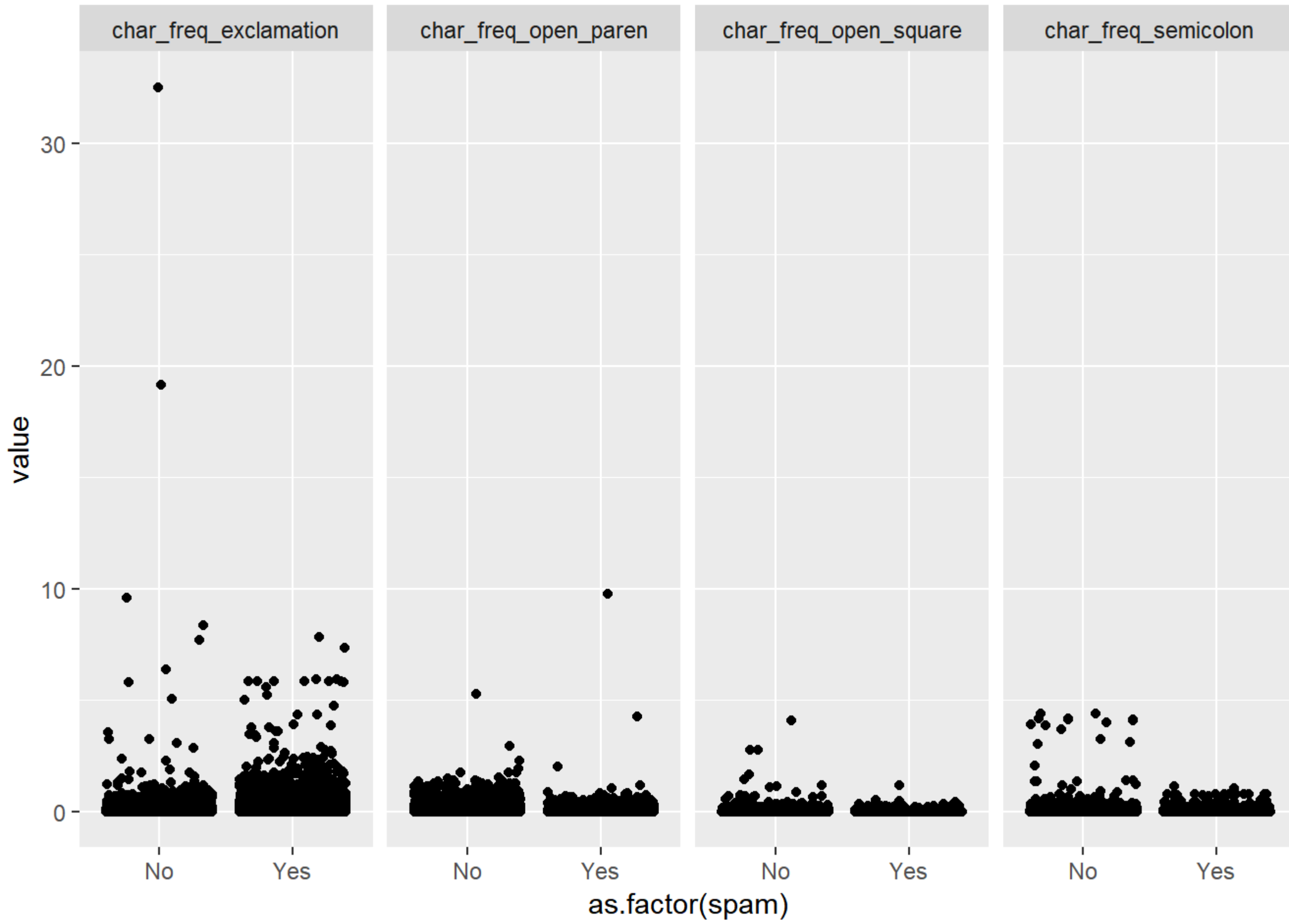
Columns 45:48

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(45:48, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of word frequency values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



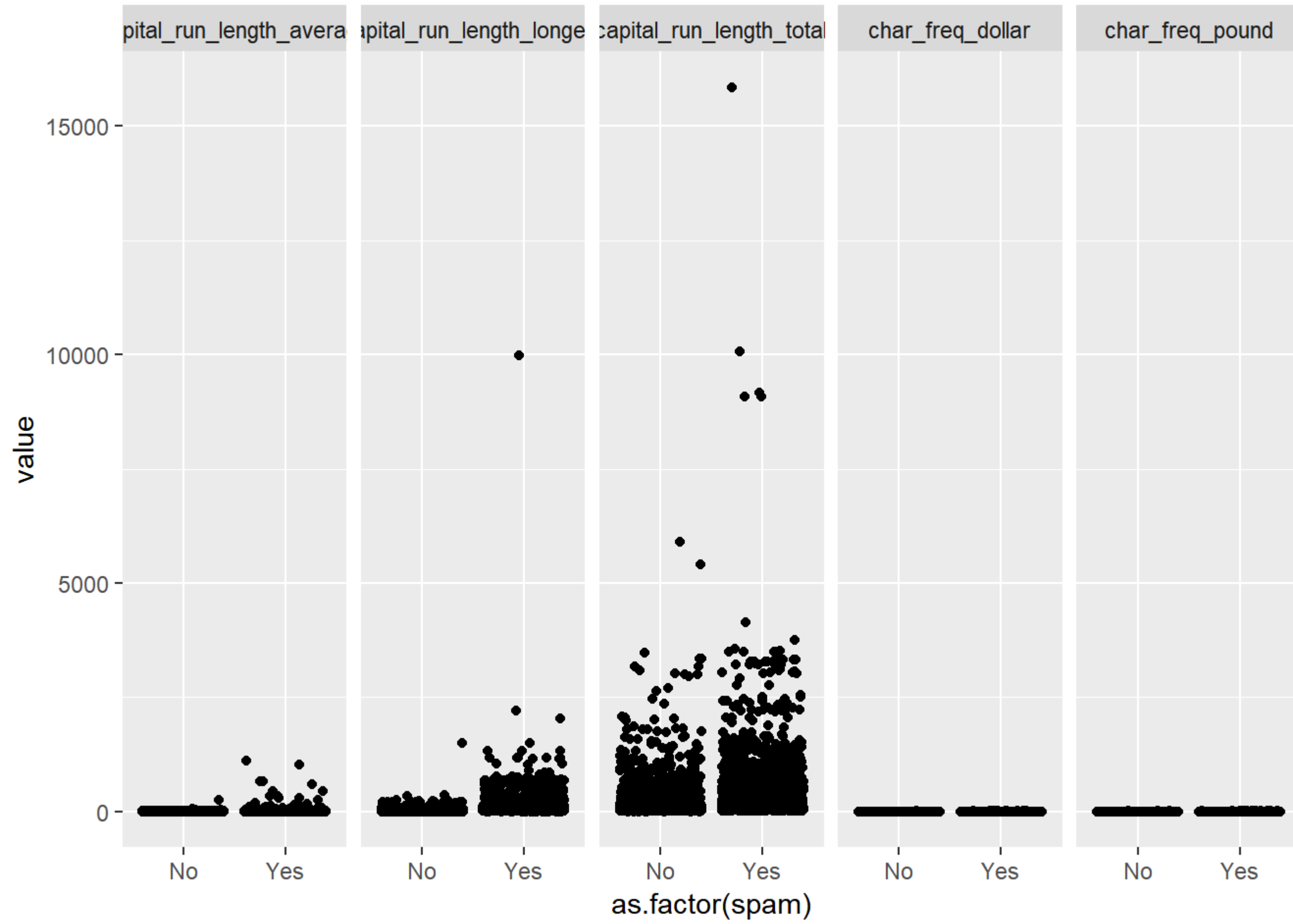
Columns 49:52

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(49:52, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of variable values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```



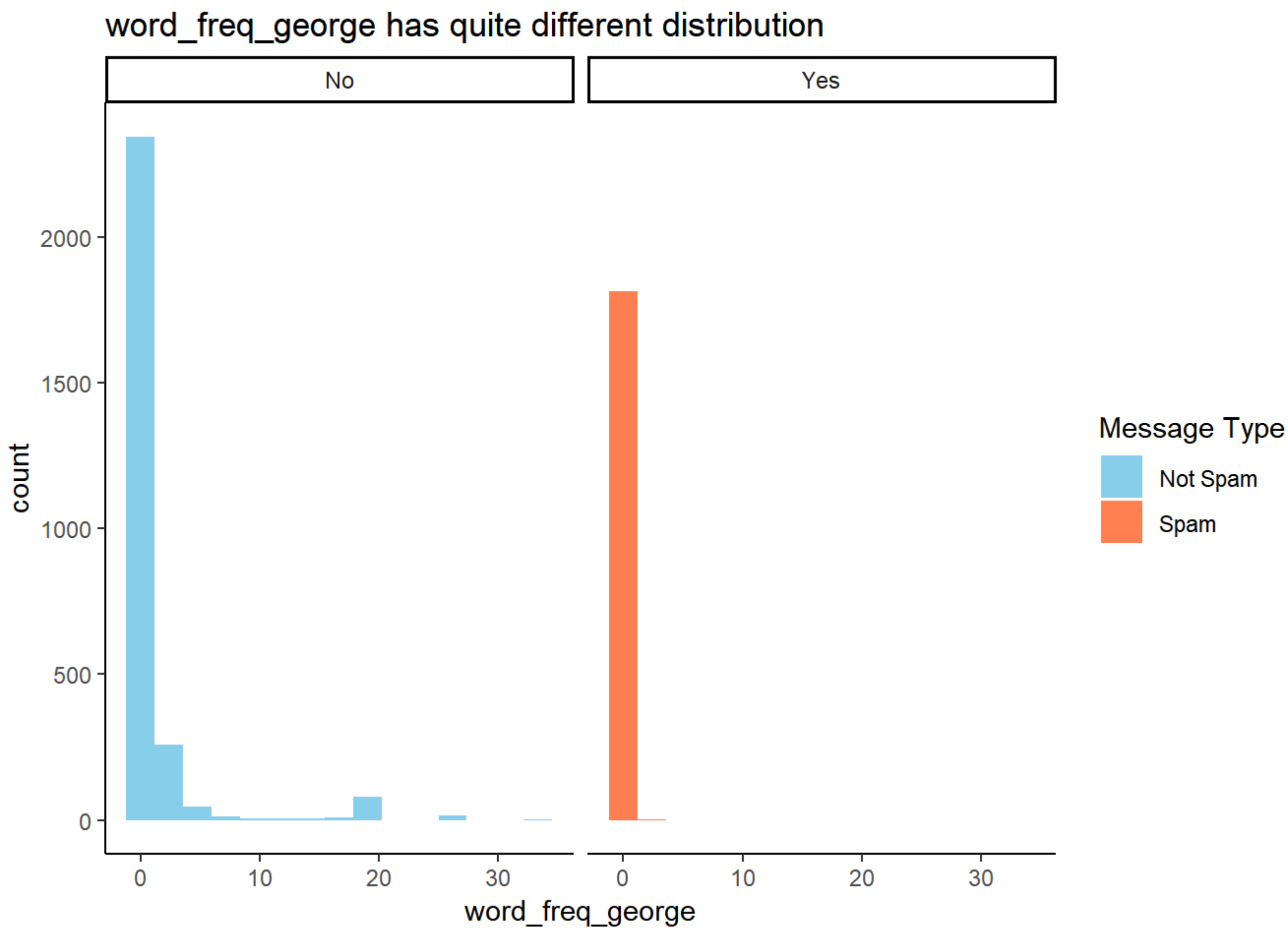
Columns 53:57

```
# Reshape spambase so that the variable names are in a column, for plotting purposes
ss = subset(spambase, select = c(53:57, spam)) %>%
  gather(key = variable_name, value=value, -spam)

# Generate jitter plots of variable values, grouped by class
ggplot(data = ss, aes(x = as.factor(spam), y = value)) +
  geom_jitter() + facet_grid(~variable_name)
```

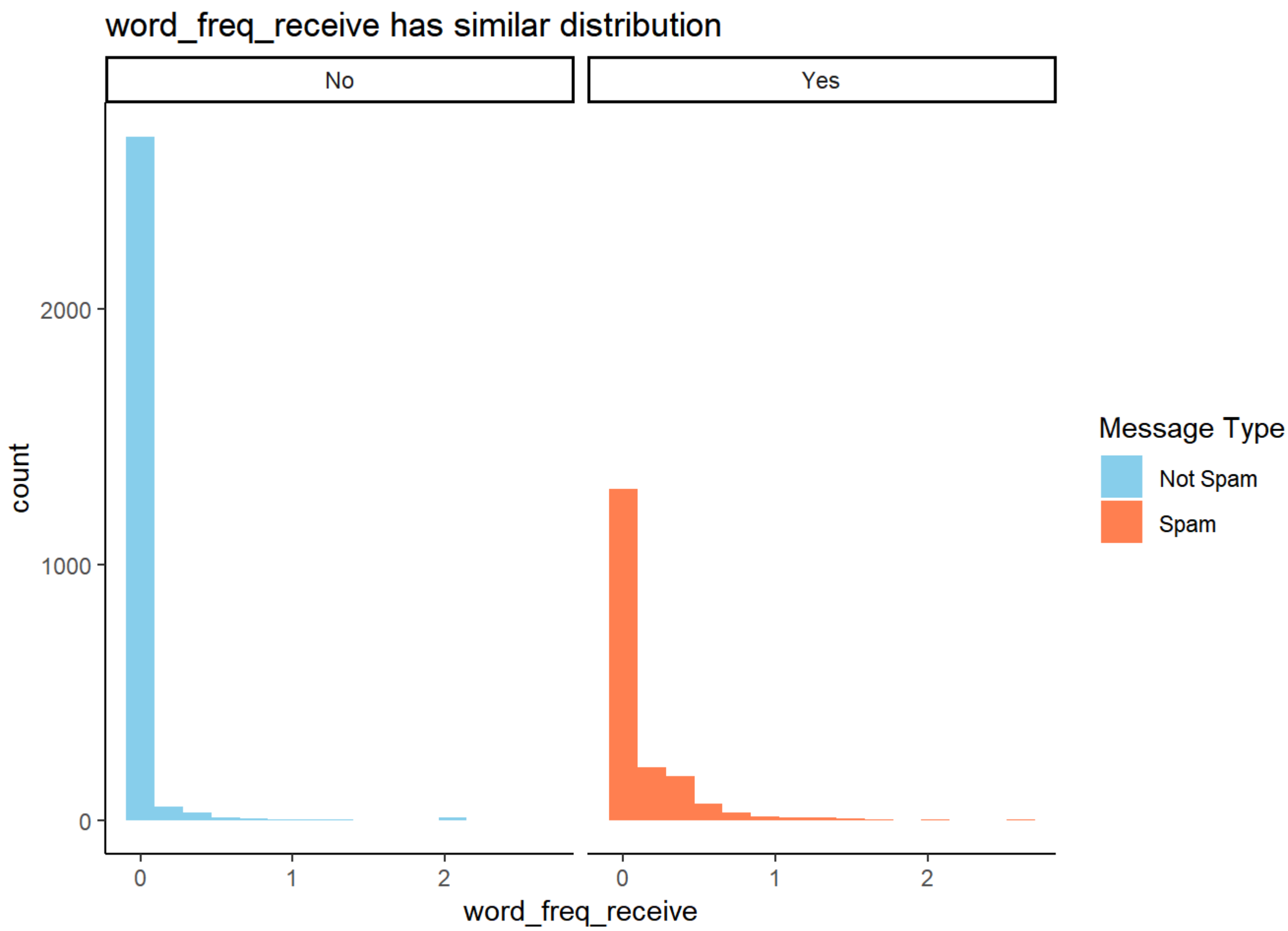


Nice plots for the report

```
ggplot(
  data = spambase,
  aes(x = word_freq_george, fill = spam)
) +
  geom_histogram(bins = 15) +
  facet_grid(~spam) +
  theme_classic() +
  scale_fill_manual("Message Type",
                    values = c("#87CEEB", "#FF7F50"),
                    labels = c("Not Spam", "Spam")) +
  ggtitle("word_freq_george has quite different distribution")
```

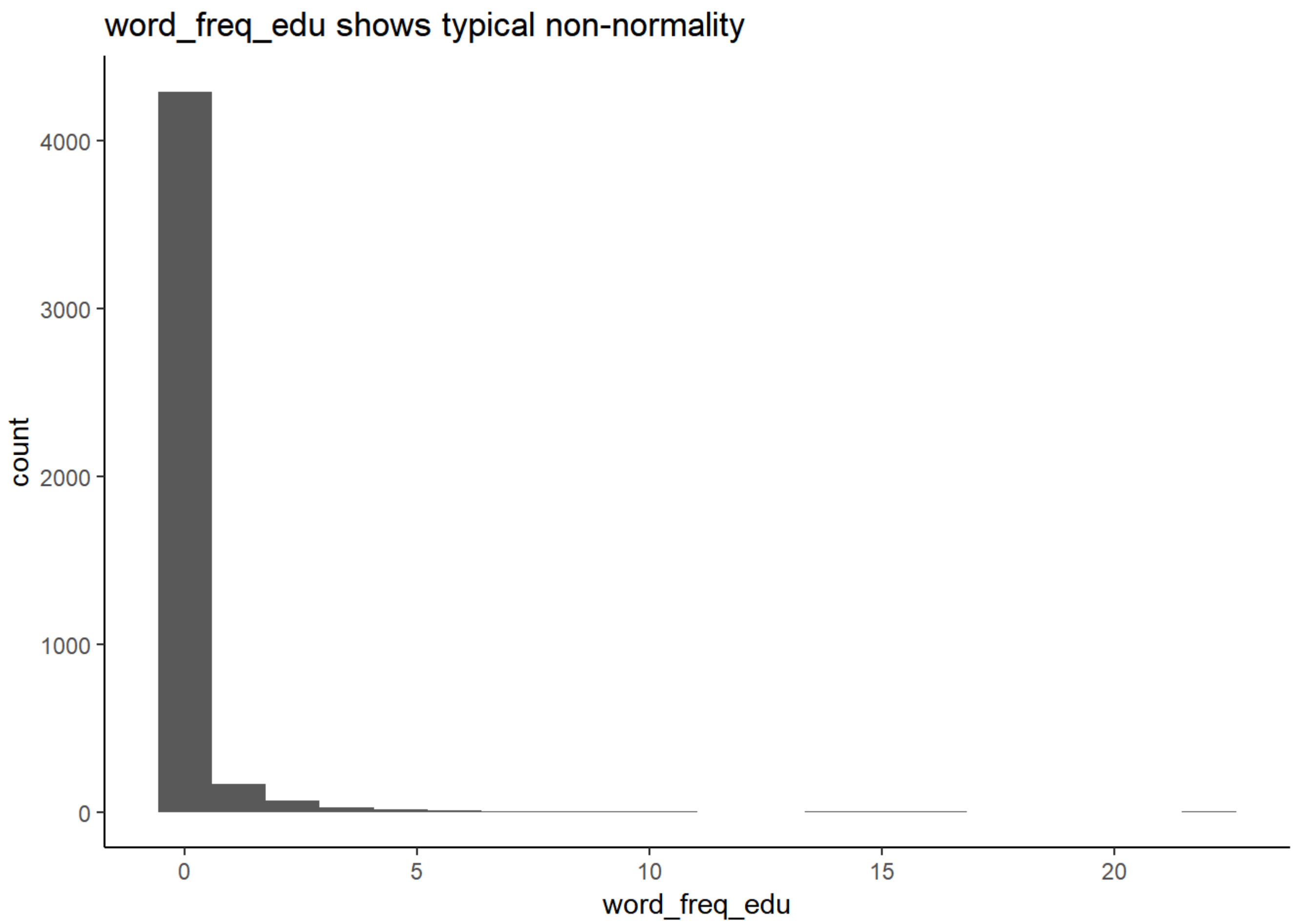



```
ggplot(
  data = spambase,
  aes(x = word_freq_receive, fill = spam)
) +
  geom_histogram(bins = 15) +
  facet_grid(~spam) +
  theme_classic() +
  scale_fill_manual("Message Type",
                    values = c("#87CEEB", "#FF7F50"),
                    labels = c("Not Spam", "Spam")) +
  ggtitle("word_freq_receive has similar distribution")
```



Not normal:

```
ggplot(
  data = spambase,
  aes(x = word_freq_edu)
) +
  geom_histogram(bins = 20) +
  #facet_grid(~spam) +
  theme_classic() +
  #scale_fill_manual("Message Type",
  #                   values = c("#87CEEB", "#FF7F50"),
  #                   labels = c("Not Spam", "Spam")) +
  ggtitle("word_freq_edu shows typical non-normality")
```

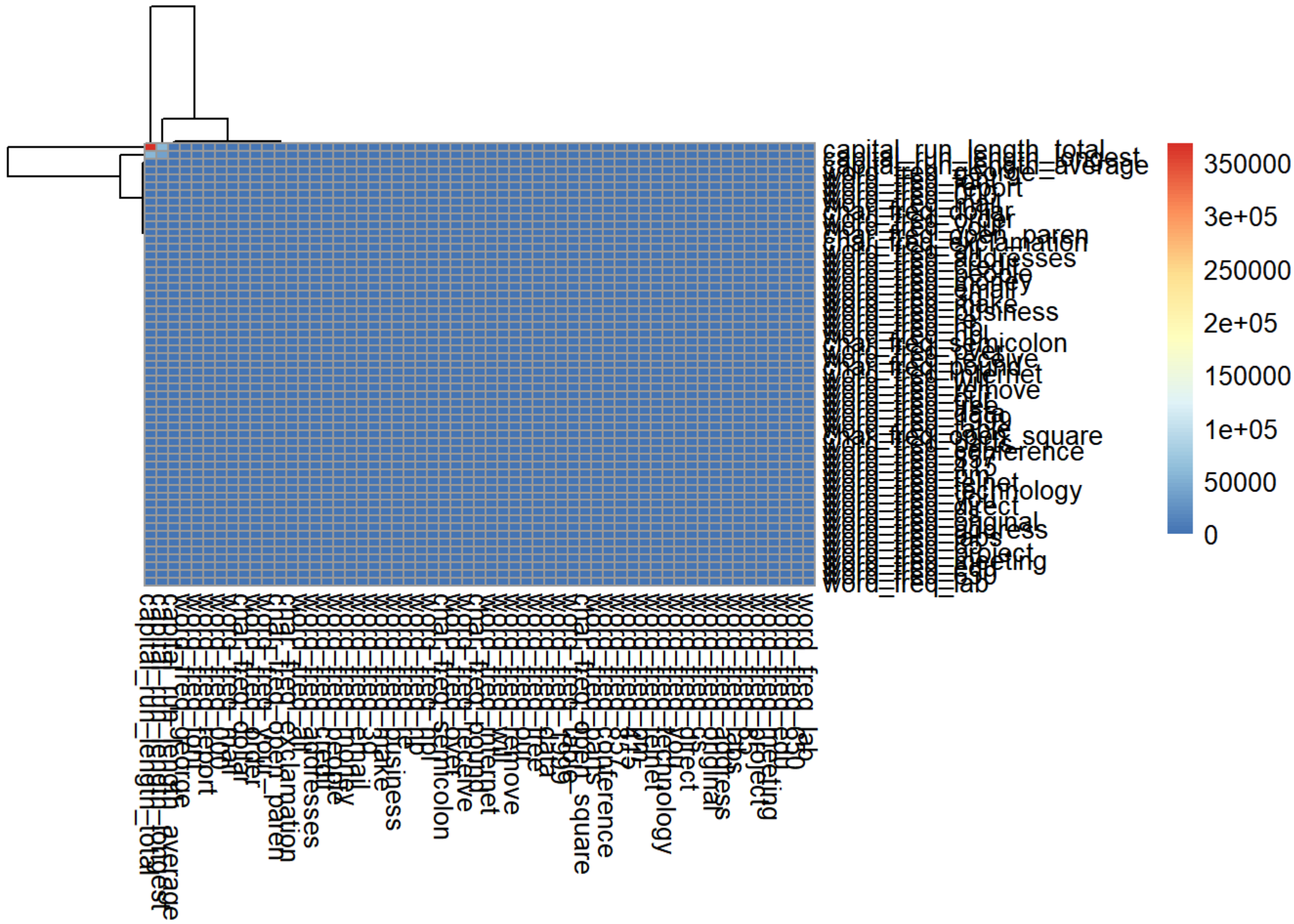



```
max(spambase$word_freq_edu)
```

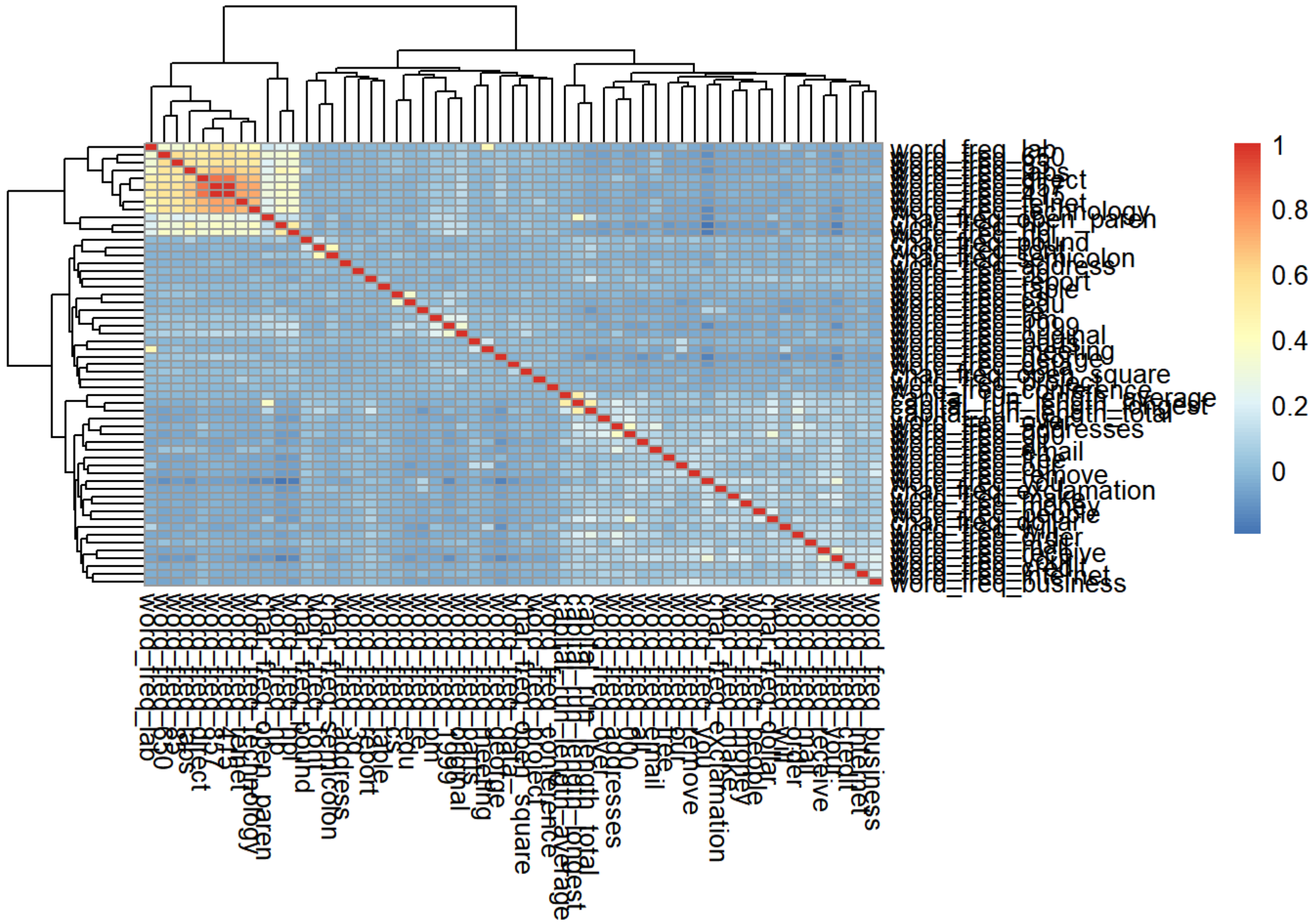
```
## [1] 22.05
```

Look at covariance and correlation

```
covariances <- data.frame(cov(spambase[,1:57]))
pheatmap::pheatmap(covariances)
```




```
#wow, we do NOT have even covariance at all.  We will need to scale, for LDA

#how does the correlation look?
correlations <- data.frame(cor(spambase[1:57]))
pheatmap::pheatmap(correlations)
```



Graphs are very simlar with a “hotspot” of correlation and covariance in the top-left corner. Let’s try to zoom in on anything over 0.5 for correlations

```
#minimum correlation?
min(correlations)
```

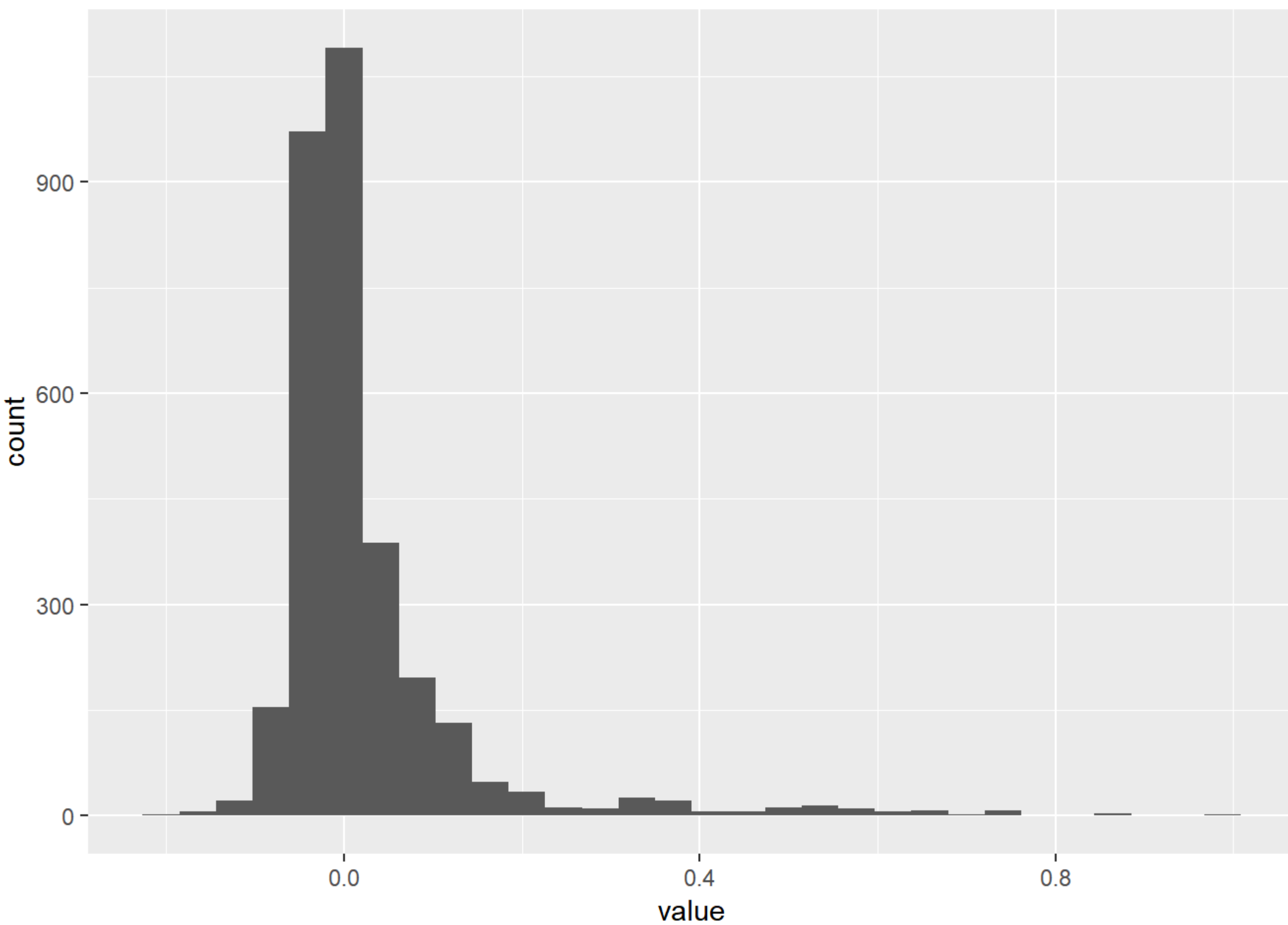
```
## [1] -0.1976832
```

```
#summary of correlations?
gathered_correlations <- correlations %>% gather()
summary(gathered_correlations)
```

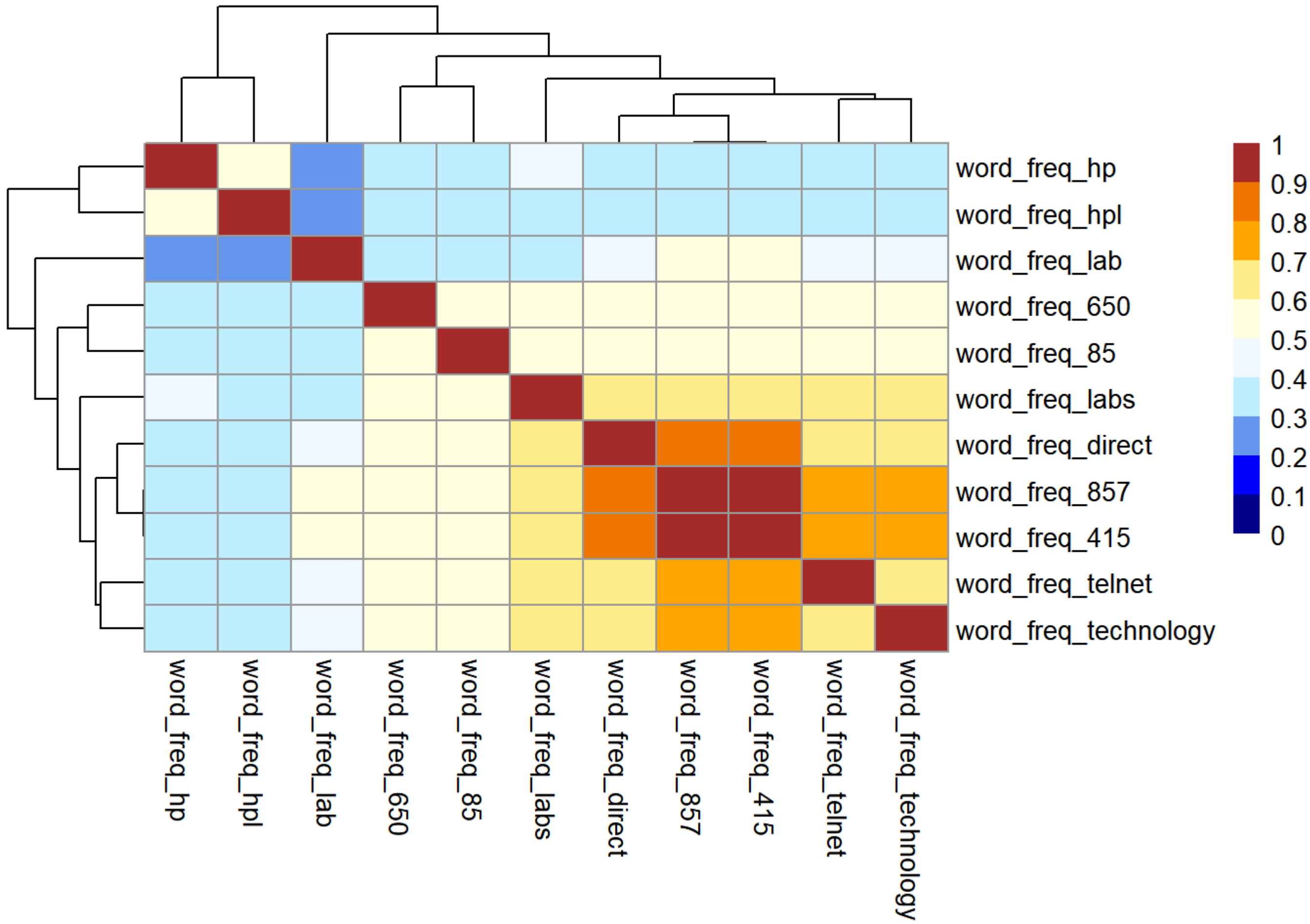
```
##      key                value
##  Length:3249        Min.   :-0.197683
##  Class :character   1st Qu.:-0.032065
##  Mode  :character   Median :-0.008344
##                     Mean   : 0.039906
##                     3rd Qu.: 0.038570
##                     Max.   : 1.000000
```

```r
#Look at the range of correlation values
ggplot(
  data = gathered_correlations[gathered_correlations$value < 1,],
  aes(x = value)
) + geom_histogram()
```

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



```
#great, we don't need to look for big negative numbers

#loop to find variable-names with "high" correlations
column_vector <- as.character()

for (i in 1:57) {
  for(j in i:57) {
    if(abs(correlations[i,j]) > 0.5 & abs(correlations[i,j]) < 1) {
      column_vector <- c(column_vector,
                         rownames(correlations)[j],
                         colnames(correlations)[i])
    }
  }
}

#get unique variable_names
column_vector_unique <- unique(column_vector)


#reduce to data of interest
cor_df_graph <- correlations[rownames(correlations) %in% column_vector_unique,
                             colnames(correlations) %in% column_vector_unique]

#graph
pheatmap::pheatmap(cor_df_graph,
                   labels_row = colnames(cor_df_graph),
                   labels_col = colnames(cor_df_graph),
                   breaks = seq(0,1,0.1), legend_breaks = seq(0,1,0.1),
                   color = c("blue4", "blue", "cornflowerblue", "lightblue1", "aliceblue",
                             "lightyellow", "lightgoldenrod1", "orange", "darkorange2", "brown"
))
```



Six variables appears to have prety high correlations. Most of the data, as we saw before, is much closer to 0. So it seems like we should scale the data to constrain the covariances in hopes of getting LDA to work better. (After scaling, covariance and correlation will come out as the same value because the stdev is defined now as 1)

Make scaled data set

```
# Make scaled data set
spambase_scaled <- as.data.frame(spambase[, 1:57] %>% scale())
spambase_scaled$spam <- spambase$spam
```

```
summary(spambase_scaled)
```

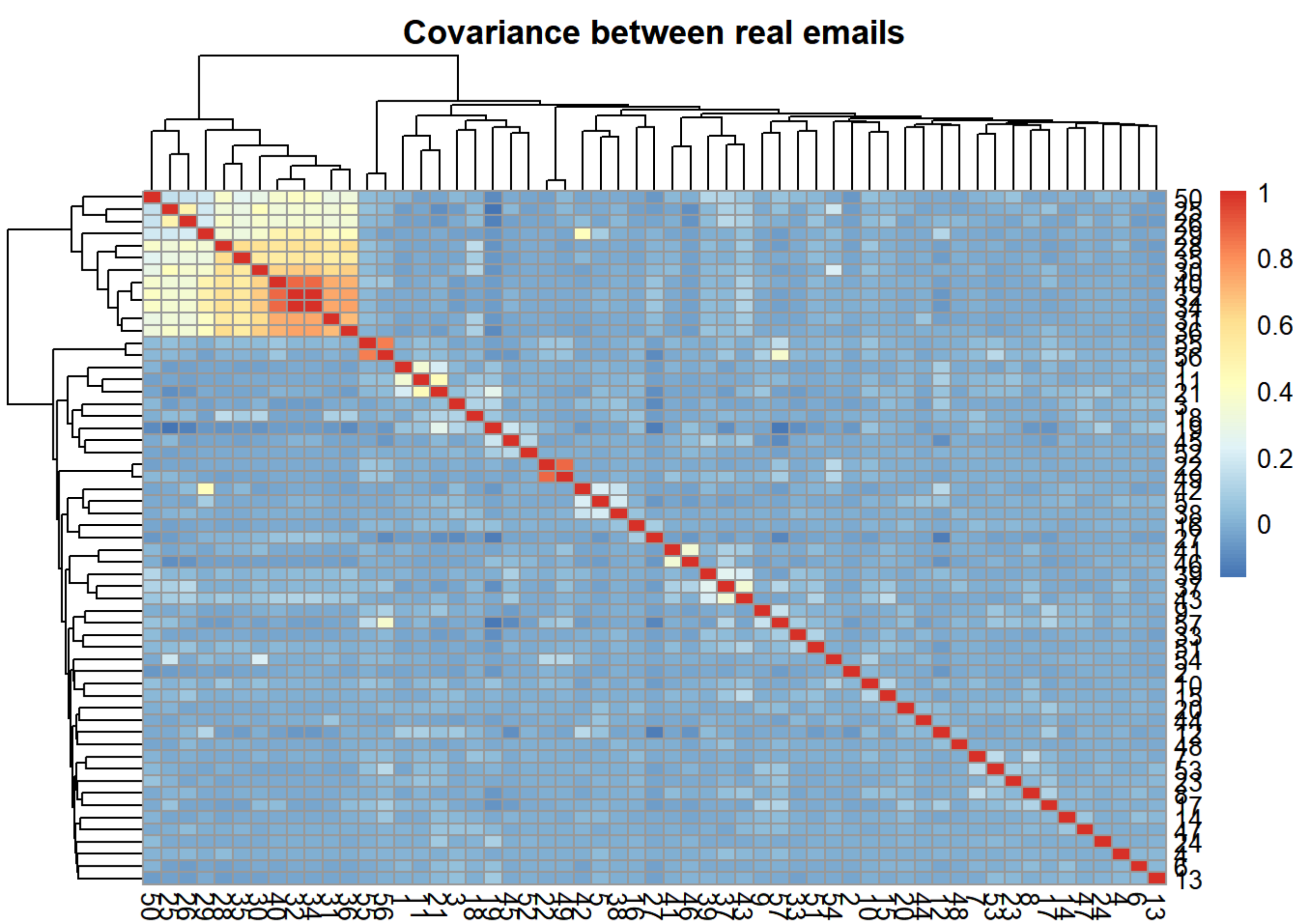
```
##  word_freq_make    word_freq_address word_freq_all      word_freq_3d
##  Min.   :-0.3424   Min.   :-0.1651   Min.   :-0.5567   Min.   :-0.04689
##  1st Qu.:-0.3424   1st Qu.:-0.1651   1st Qu.:-0.5567   1st Qu.:-0.04689
##  Median :-0.3424   Median :-0.1651   Median :-0.5567   Median :-0.04689
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.:-0.3424   3rd Qu.:-0.1651   3rd Qu.: 0.2764   3rd Qu.:-0.04689
##  Max.   :14.5254   Max.   :10.8998   Max.   : 9.5595   Max.   :30.63795
##  word_freq_our     word_freq_over    word_freq_remove  word_freq_internet
##  Min.   :-0.4643   Min.   :-0.3502   Min.   :-0.2918   Min.   :-0.2625
##  1st Qu.:-0.4643   1st Qu.:-0.3502   1st Qu.:-0.2918   1st Qu.:-0.2625
##  Median :-0.4643   Median :-0.3502   Median :-0.2918   Median :-0.2625
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.1008   3rd Qu.:-0.3502   3rd Qu.:-0.2918   3rd Qu.:-0.2625
##  Max.   :14.4053   Max.   :21.1234   Max.   :18.2806   Max.   :27.4383
##  word_freq_order   word_freq_mail    word_freq_receive word_freq_will
##  Min.   :-0.3233   Min.   :-0.3713   Min.   :-0.2968   Min.   :-0.6286
##  1st Qu.:-0.3233   1st Qu.:-0.3713   1st Qu.:-0.2968   1st Qu.:-0.6286
##  Median :-0.3233   Median :-0.3713   Median :-0.2968   Median :-0.5126
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.:-0.3233   3rd Qu.:-0.1232   3rd Qu.:-0.2968   3rd Qu.: 0.2998
##  Max.   :18.5558   Max.   :27.8254   Max.   :12.6532   Max.   :10.5934
##  word_freq_people word_freq_report  word_freq_addresses word_freq_free
##  Min.   :-0.312   Min.   :-0.1749   Min.   :-0.1901     Min.   :-0.3013
##  1st Qu.:-0.312   1st Qu.:-0.1749   1st Qu.:-0.1901     1st Qu.:-0.3013
##  Median :-0.312   Median :-0.1749   Median :-0.1901     Median :-0.3013
##  Mean   : 0.000   Mean   : 0.0000   Mean   : 0.0000     Mean   : 0.0000
##  3rd Qu.:-0.312   3rd Qu.:-0.1749   3rd Qu.:-0.1901     3rd Qu.:-0.1802
##  Max.   :18.124   Max.   :29.6595   Max.   :16.8472     Max.   :23.9178
##  word_freq_business word_freq_email   word_freq_you     word_freq_credit
##  Min.   :-0.3211    Min.   :-0.3478   Min.   :-0.9361   Min.   :-0.1679
##  1st Qu.:-0.3211    1st Qu.:-0.3478   1st Qu.:-0.9361   1st Qu.:-0.1679
##  Median :-0.3211    Median :-0.3478   Median :-0.1983   Median :-0.1679
##  Mean   : 0.0000    Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.:-0.3211    3rd Qu.:-0.3478   3rd Qu.: 0.5508   3rd Qu.:-0.1679
##  Max.   :15.7580    Max.   :16.7669   Max.   : 9.6244   Max.   :35.4955
##  word_freq_your    word_freq_font    word_freq_000     word_freq_money
##  Min.   :-0.6743   Min.   :-0.1182   Min.   :-0.2902   Min.   :-0.213
##  1st Qu.:-0.6743   1st Qu.:-0.1182   1st Qu.:-0.2902   1st Qu.:-0.213
##  Median :-0.4911   Median :-0.1182   Median :-0.2902   Median :-0.213
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.000
##  3rd Qu.: 0.3833   3rd Qu.:-0.1182   3rd Qu.:-0.2902   3rd Qu.:-0.213
##  Max.   : 8.5777   Max.   :16.5525   Max.   :15.2685   Max.   :28.027
##   word_freq_hp     word_freq_hpl     word_freq_george  word_freq_650
##  Min.   :-0.3288   Min.   :-0.2992   Min.   :-0.2279   Min.   :-0.2318
##  1st Qu.:-0.3288   1st Qu.:-0.2992   1st Qu.:-0.2279   1st Qu.:-0.2318
##  Median :-0.3288   Median :-0.2992   Median :-0.2279   Median :-0.2318
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.:-0.3288   3rd Qu.:-0.2992   3rd Qu.:-0.2279   3rd Qu.:-0.2318
##  Max.   :12.1342   Max.   :18.4842   Max.   : 9.6703   Max.   :16.6460
##  word_freq_lab     word_freq_labs    word_freq_telnet  word_freq_857
##  Min.   :-0.1667   Min.   :-0.2252   Min.   :-0.1605   Min.   :-0.1432
##  1st Qu.:-0.1667   1st Qu.:-0.2252   1st Qu.:-0.1605   1st Qu.:-0.1432
##  Median :-0.1667   Median :-0.2252   Median :-0.1605   Median :-0.1432
```

```
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.:-0.1667   3rd Qu.:-0.2252   3rd Qu.:-0.1605   3rd Qu.:-0.1432  
##  Max.   :23.9010   Max.   :12.6503   Max.   :30.8267   Max.   :14.3443  
##  word_freq_data    word_freq_415      word_freq_85     
##  Min.   :-0.1749   Min.   :-0.1452   Min.   :-0.1981  
##  1st Qu.:-0.1749   1st Qu.:-0.1452   1st Qu.:-0.1981  
##  Median :-0.1749   Median :-0.1452   Median :-0.1981  
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.:-0.1749   3rd Qu.:-0.1452   3rd Qu.:-0.1981  
##  Max.   :32.5284   Max.   :14.3033   Max.   :37.3776  
##  word_freq_technology word_freq_1999    word_freq_parts   
##  Min.   :-0.2421      Min.   :-0.3234   Min.   :-0.05983  
##  1st Qu.:-0.2421      1st Qu.:-0.3234   1st Qu.:-0.05983  
##  Median :-0.2421      Median :-0.3234   Median :-0.05983  
##  Mean   : 0.0000      Mean   : 0.0000   Mean   : 0.00000  
##  3rd Qu.:-0.2421      3rd Qu.:-0.3234   3rd Qu.:-0.05983  
##  Max.   :18.8576      Max.   :15.9476   Max.   :37.69213  
##   word_freq_pm     word_freq_direct   word_freq_cs     word_freq_meeting
##  Min.   :-0.1809   Min.   :-0.1853   Min.   :-0.1209   Min.   :-0.1726  
##  1st Qu.:-0.1809   1st Qu.:-0.1853   1st Qu.:-0.1209   1st Qu.:-0.1726  
##  Median :-0.1809   Median :-0.1853   Median :-0.1209   Median :-0.1726  
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.:-0.1809   3rd Qu.:-0.1853   3rd Qu.:-0.1209   3rd Qu.:-0.1726  
##  Max.   :25.3786   Max.   :13.4180   Max.   :19.6463   Max.   :18.4498  
##  word_freq_original word_freq_project  word_freq_re     word_freq_edu    
##  Min.   :-0.206     Min.   :-0.1273   Min.   :-0.2977   Min.   :-0.1974  
##  1st Qu.:-0.206     1st Qu.:-0.1273   1st Qu.:-0.2977   1st Qu.:-0.1974  
##  Median :-0.206     Median :-0.1273   Median :-0.2977   Median :-0.1974  
##  Mean   : 0.000     Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
##  3rd Qu.:-0.206     3rd Qu.:-0.1273   3rd Qu.:-0.1890   3rd Qu.:-0.1974  
##  Max.   :15.745     Max.   :32.0283   Max.   :20.8748   Max.   :24.0036  
##  word_freq_table    word_freq_conference char_freq_semicolon
##  Min.   :-0.07138   Min.   :-0.1115      Min.   :-0.1584    
##  1st Qu.:-0.07138   1st Qu.:-0.1115      1st Qu.:-0.1584    
##  Median :-0.07138   Median :-0.1115      Median :-0.1584    
##  Mean   : 0.00000   Mean   : 0.0000      Mean   : 0.0000    
##  3rd Qu.:-0.07138   3rd Qu.:-0.1115      3rd Qu.:-0.1584    
##  Max.   :28.37858   Max.   :34.8860      Max.   :17.8519    
##  char_freq_open_paren char_freq_open_square char_freq_exclamation
##  Min.   :-0.5142      Min.   :-0.1552       Min.   :-0.32988     
##  1st Qu.:-0.5142      1st Qu.:-0.1552       1st Qu.:-0.32988     
##  Median :-0.2738      Median :-0.1552       Median :-0.32988     
##  Mean   : 0.0000      Mean   : 0.0000       Mean   : 0.00000     
##  3rd Qu.: 0.1811      3rd Qu.:-0.1552       3rd Qu.: 0.05631     
##  Max.   :35.5568      Max.   :37.1503       Max.   :39.48762     
##  char_freq_dollar   char_freq_pound  capital_run_length_average
##  Min.   :-0.30832   Min.   :-0.103   Min.   :-0.13210          
##  1st Qu.:-0.30832   1st Qu.:-0.103   1st Qu.:-0.11357          
##  Median :-0.30832   Median :-0.103   Median :-0.09189          
##  Mean   : 0.00000   Mean   : 0.000   Mean   : 0.00000          
##  3rd Qu.:-0.09684   3rd Qu.:-0.103   3rd Qu.:-0.04682          
##  Max.   :24.10583   Max.   :46.082   Max.   :34.58328          
##  capital_run_length_longest capital_run_length_total  spam     
##  Min.   :-0.26257           Min.   :-0.46556         No :2788  
```

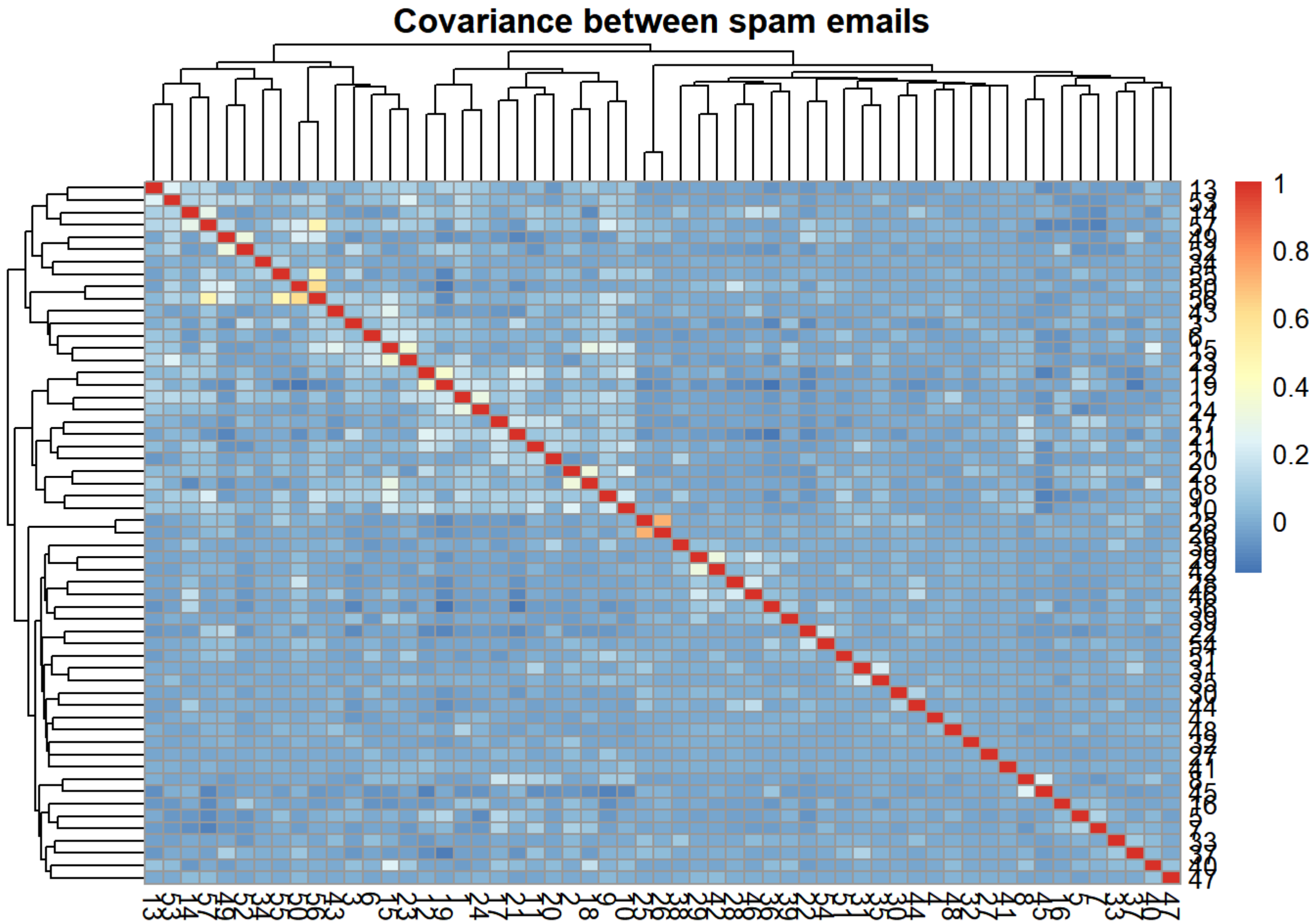
```
##  1st Qu.:-0.23692          1st Qu.:-0.40948        Yes:1813
##  Median :-0.19074          Median :-0.31053
##  Mean   : 0.00000          Mean   : 0.00000
##  3rd Qu.:-0.04707          3rd Qu.:-0.02851
##  Max.   :50.98651          Max.   :25.65806
```

Covariances between classes, with scaled dataset

```
#separate data into two groups: spam and not spam
spam_real <- spambase_scaled[spambase$spam == "No",1:57]
spam_spam <- spambase_scaled[spambase$spam == "Yes",1:57]

#compare general covariance values between them (cov now equal to cor)
#change labels to index numbers so the picture is bigger
pheatmap::pheatmap(cor(spam_real),
                   labels_row = seq(1:57),
                   labels_col = seq(1:57),
                   main = "Covariance between real emails")
```




```
pheatmap::pheatmap(cor(spam_spam),
                   labels_row = seq(1:57),
                   labels_col = seq(1:57),
                   main = "Covariance between spam emails")
```



The real messages have a wider range of covariance than the spam message, even with scaling. This suggests that LDA will suffer.

Unoptimized models

The unoptimized models we will consider are random forests, linear discriminant analysis, quadratic discriminant analysis, and support vector machines.

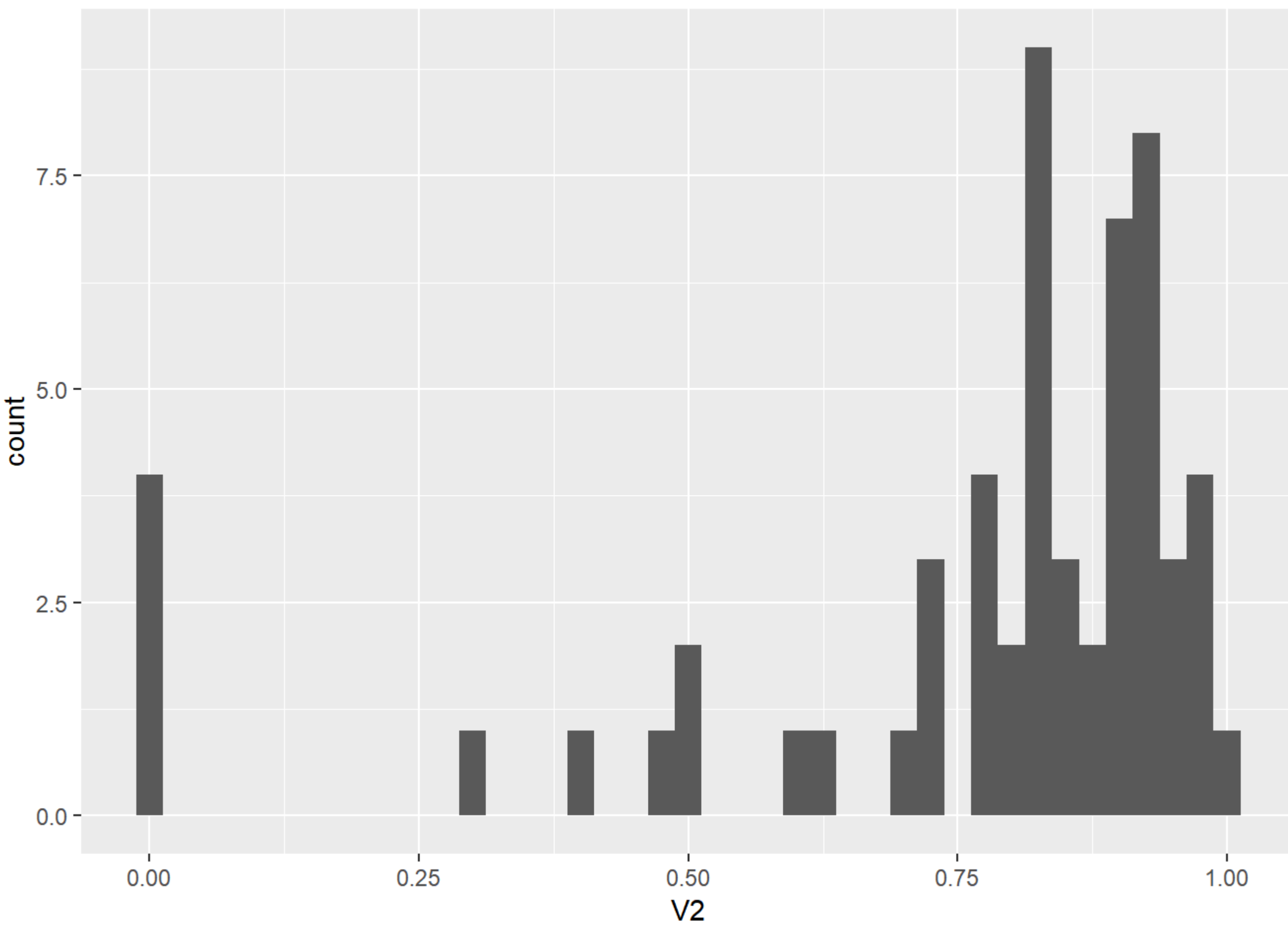
In the code below, QDA gave a rank-deficiency error the first time it was run. So we investigate and fix it here. There are many zero entries:

```
# Calculate the frequency of zero entries in each column
zero_frequency <- as.data.frame(matrix(data = 0,
                                       nrow = ncol(spambase),
                                       ncol = 2))
for(i in 1:58) {
  zero_frequency[i,2] <- sum(spambase[[i]] == 0)/4601
  zero_frequency[i,1] <- colnames(spambase[i])
}

# Print a sorted table in descending order
zero_frequency %>% arrange(desc(V2))
```

```
##                        V1        V2
## 1             word_freq_3d 0.9897848
## 2          word_freq_table 0.9863073
## 3          word_freq_parts 0.9819604
## 4           word_freq_font 0.9745707
## 5             word_freq_cs 0.9678331
## 6     word_freq_conference 0.9558792
## 7            word_freq_857 0.9554445
## 8            word_freq_415 0.9532710
## 9         word_freq_telnet 0.9363182
## 10       word_freq_project 0.9289285
## 11     word_freq_addresses 0.9269724
## 12       word_freq_meeting 0.9258857
## 13        word_freq_report 0.9224082
## 14           word_freq_lab 0.9191480
## 15      word_freq_original 0.9184960
## 16            word_freq_pm 0.9165399
## 17          word_freq_data 0.9119757
## 18        word_freq_credit 0.9078461
## 19        word_freq_direct 0.9015431
## 20           word_freq_650 0.8993697
## 21          word_freq_labs 0.8980656
## 22            word_freq_85 0.8945881
## 23           word_freq_edu 0.8876331
## 24    char_freq_open_square 0.8850250
## 25    word_freq_technology 0.8698109
## 26           word_freq_000 0.8524234
## 27       word_freq_receive 0.8459031
## 28         word_freq_money 0.8402521
## 29         char_freq_pound 0.8369920
## 30         word_freq_order 0.8319930
## 31        word_freq_george 0.8304716
## 32     char_freq_semicolon 0.8282982
## 33        word_freq_remove 0.8246033
## 34           word_freq_hpl 0.8237340
## 35      word_freq_internet 0.8209085
## 36          word_freq_1999 0.8198218
## 37        word_freq_people 0.8148229
## 38       word_freq_address 0.8048250
## 39      word_freq_business 0.7906977
## 40          word_freq_over 0.7828733
## 41         word_freq_email 0.7743969
## 42          word_freq_make 0.7711367
## 43            word_freq_hp 0.7630950
## 44          word_freq_free 0.7302760
## 45          word_freq_mail 0.7170180
## 46            word_freq_re 0.7150619
## 47        char_freq_dollar 0.6957183
## 48           word_freq_our 0.6200826
## 49           word_freq_all 0.5896544
## 50    char_freq_exclamation 0.5092371
## 51          word_freq_will 0.4946751
## 52          word_freq_your 0.4733754
```

```
## 53       char_freq_open_paren 0.4099109
## 54               word_freq_you 0.2986307
## 55 capital_run_length_average 0.0000000
## 56 capital_run_length_longest 0.0000000
## 57   capital_run_length_total 0.0000000
## 58                       spam 0.0000000
```

```
ggplot(
  data = zero_frequency,
  aes(x = V2)
) + geom_histogram(binwidth = 0.025)
```

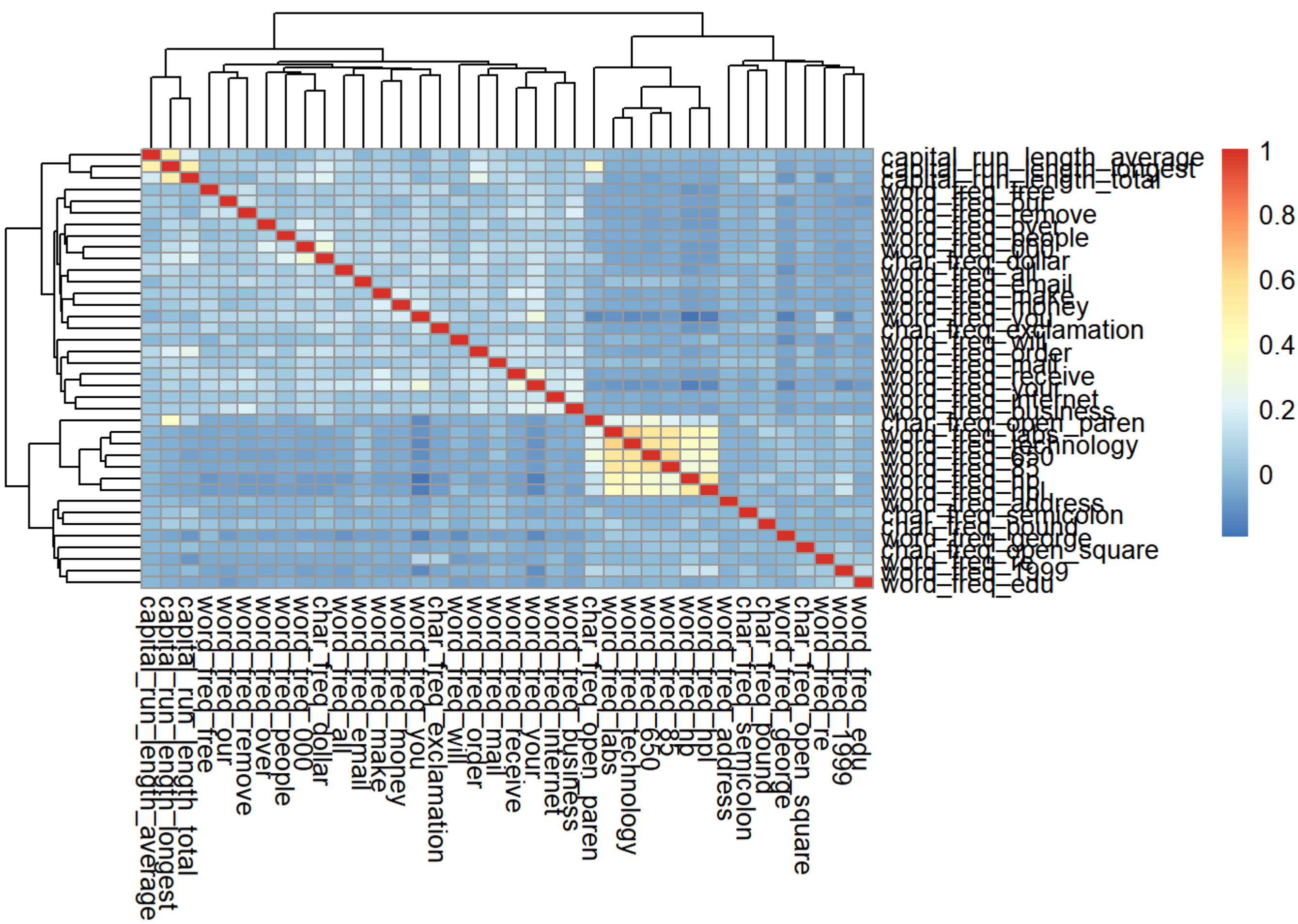


We will remove variables that are zero for over 90% of the observations for the QDA calculations (removing variables that are zero for over 95% of the observations still occasionally threw a rank-deficiency error).

```
# Remove low-variability columns to make data set for QDA
spambase_qda <- spambase %>%
  dplyr::select(-word_freq_3d, -word_freq_table, -word_freq_parts,
                -word_freq_font, -word_freq_cs, -word_freq_conference,
                -word_freq_857, -word_freq_415, -word_freq_telnet,
                -word_freq_project, -word_freq_addresses, -word_freq_meeting,
                -word_freq_report, -word_freq_lab, -word_freq_original,
                -word_freq_pm, -word_freq_data, -word_freq_credit,
                -word_freq_direct)

# Make scaled data set for QDA
spambase_scaled_qda <- as.data.frame(spambase_qda[, 1:38] %>% scale())
spambase_scaled_qda$spam <- spambase_qda$spam
```

Check correlation (same as covariance) here also

```
correlation_qda <- data.frame(cor(spambase_scaled_qda[,1:38]))
pheatmap::pheatmap(correlation_qda)
```



Interesting that the "highly correlated" data is less highly correlated but still part of the dataset. I would have thought those might be the same as the excluded variables.

Now: run the base models without error

Set up basic parameters

```
R = 50 # set the number of replications

# set up train control to do CV
#tunegrid = expand.grid(mtry=c(6:10))

fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

# create the error matrix to store values
err_mat = matrix(0, ncol=4, nrow=R)

# create sensitivity matrix to store values
sens_mat = matrix(0, ncol=4, nrow=R)

# create specificity matrix to store values
spec_mat = matrix(0, ncol=4, nrow=R)
```

Try the scaled data

```
R = 50 # set the number of replications

# set up train control to do CV
#tunegrid = expand.grid(mtry=c(6:10))

fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

# create the error matrix to store values
err_mat = matrix(0, ncol=4, nrow=R)

# create sensitivity matrix to store values
sens_mat = matrix(0, ncol=4, nrow=R)

# create specificity matrix to store values
spec_mat = matrix(0, ncol=4, nrow=R)
```

```r
set.seed(1)

# Loop through repetitions
for (r in 1:R){
  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Form training and test sets
  spam_train = spambase_scaled[id$tr,]
  spam_test = spambase_scaled[id$ts,]
  spam_train_qda = spambase_scaled_qda[id$tr,]
  spam_test_qda = spambase_scaled_qda[id$ts,]

  # Run random forest model
  mod_rf = randomForest(spam ~ .,
                        spam_train,
                        ntree = 100,
                        trControl = fitControl,
                        metric = "ROC")

  # Run LDA model
  mod_lda <- lda(spam ~ .,
                 spam_train,
                 trControl = fitControl,
                 metric = "ROC")

  # Run QDA model
  mod_qda <- qda(spam ~ .,
                 spam_train_qda,
                 trControl = fitControl,
                 metric = "ROC")

  # Run SVM model
  mod_svm <- svm(spam~.,
                 spam_train,
                 cross=5,
                 type='C-classification',
                 metric = "ROC")

  # Make predictions on test set
  yhat_rf = predict(mod_rf, spam_test[,-58])
  yhat_lda = predict(mod_lda, spam_test[,-58])$class
  yhat_qda = predict(mod_qda, spam_test_qda[,-39])$class
  yhat_svm = predict(mod_svm, spam_test[,-58])

  # Calculate error rates
  err_mat[r,1] = mean(yhat_rf!=spam_test[,58])
  err_mat[r,2] = mean(yhat_lda!=spam_test[,58])
  err_mat[r,3] = mean(yhat_qda!=spam_test_qda[,39])
  err_mat[r,4] = mean(yhat_svm!=spam_test[,58])
```

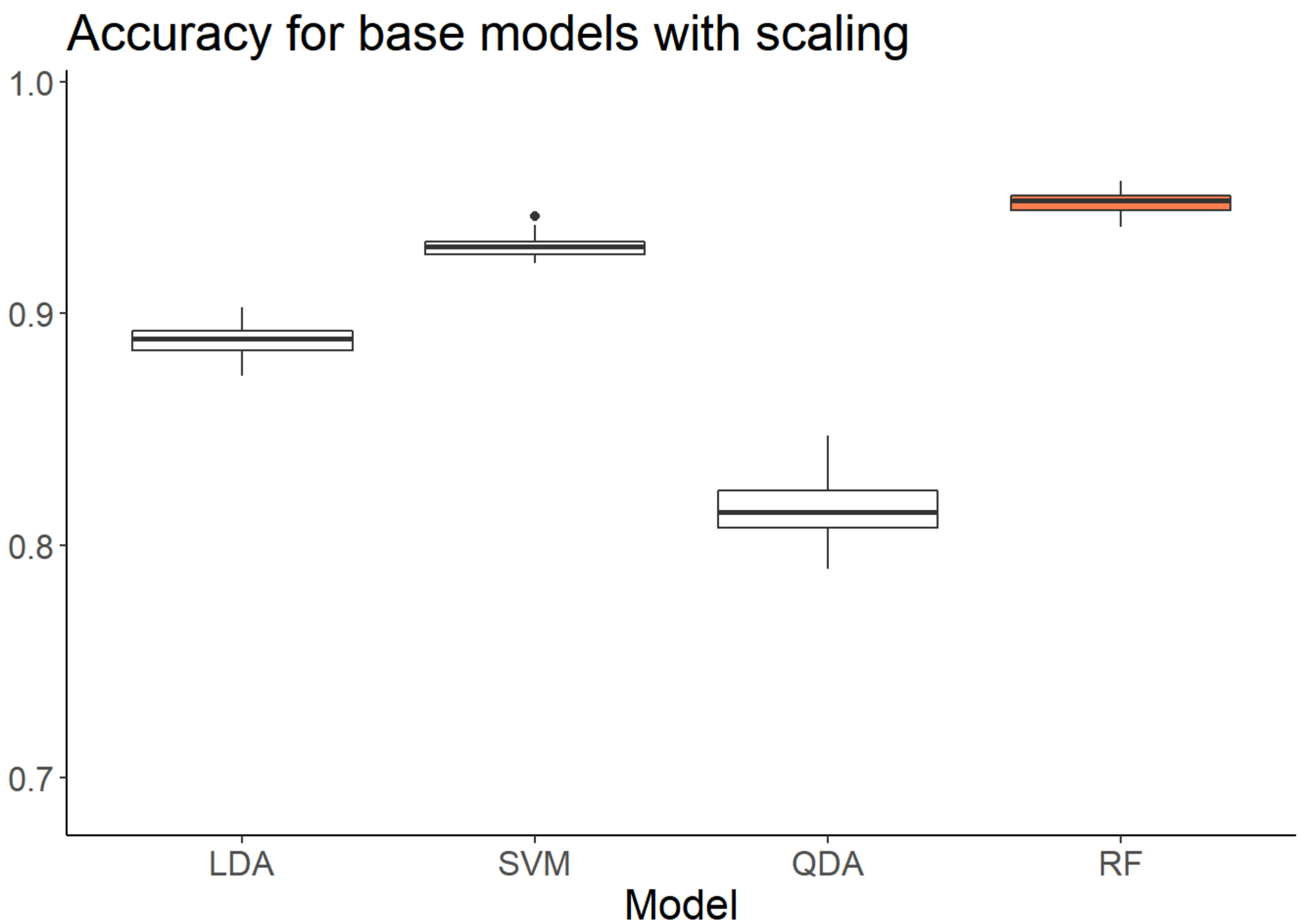
```r
  # Calculate confusion matrices
  cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")
  cm_lda <- confusionMatrix(yhat_lda, spam_test[,58], positive = "No")
  cm_qda <- confusionMatrix(yhat_qda, spam_test_qda[,39], positive = "No")
  cm_svm <- confusionMatrix(yhat_svm, spam_test[,58], positive = "No")

  # Calculate sensitivity
  sens_mat[r,1] = cm_rf$byClass["Sensitivity"]
  sens_mat[r,2] = cm_lda$byClass["Sensitivity"]
  sens_mat[r,3] = cm_qda$byClass["Sensitivity"]
  sens_mat[r,4] = cm_svm$byClass["Sensitivity"]

  # Calculate specificity
  spec_mat[r,1] = cm_rf$byClass["Specificity"]
  spec_mat[r,2] = cm_lda$byClass["Specificity"]
  spec_mat[r,3] = cm_qda$byClass["Specificity"]
  spec_mat[r,4] = cm_svm$byClass["Specificity"]

  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Name output appropriately, melt to prepare for plotting
colnames(err_mat) = c("RF", "LDA", "QDA", "SVM")
err_mat_melt = melt(as.data.frame(err_mat))
colnames(err_mat_melt) = c('Method','Error')

colnames(sens_mat) = c("RF", "LDA", "QDA", "SVM")
sens_mat_melt = melt(as.data.frame(sens_mat))
colnames(sens_mat_melt) = c('Method','Sensitivity')

colnames(spec_mat) = c("RF", "LDA", "QDA", "SVM")
spec_mat_melt = melt(as.data.frame(spec_mat))
colnames(spec_mat_melt) = c('Method','Specificity')
```

```r
# Reorder Method factor levels
err_mat_melt$Method <- ordered(err_mat_melt$Method,
                               levels = c("LDA", "SVM", "QDA", "RF"))
sens_mat_melt$Method <- ordered(sens_mat_melt$Method,
                                levels = c("LDA", "SVM", "QDA", "RF"))
spec_mat_melt$Method <- ordered(spec_mat_melt$Method,
                                levels = c("LDA", "SVM", "QDA", "RF"))
```

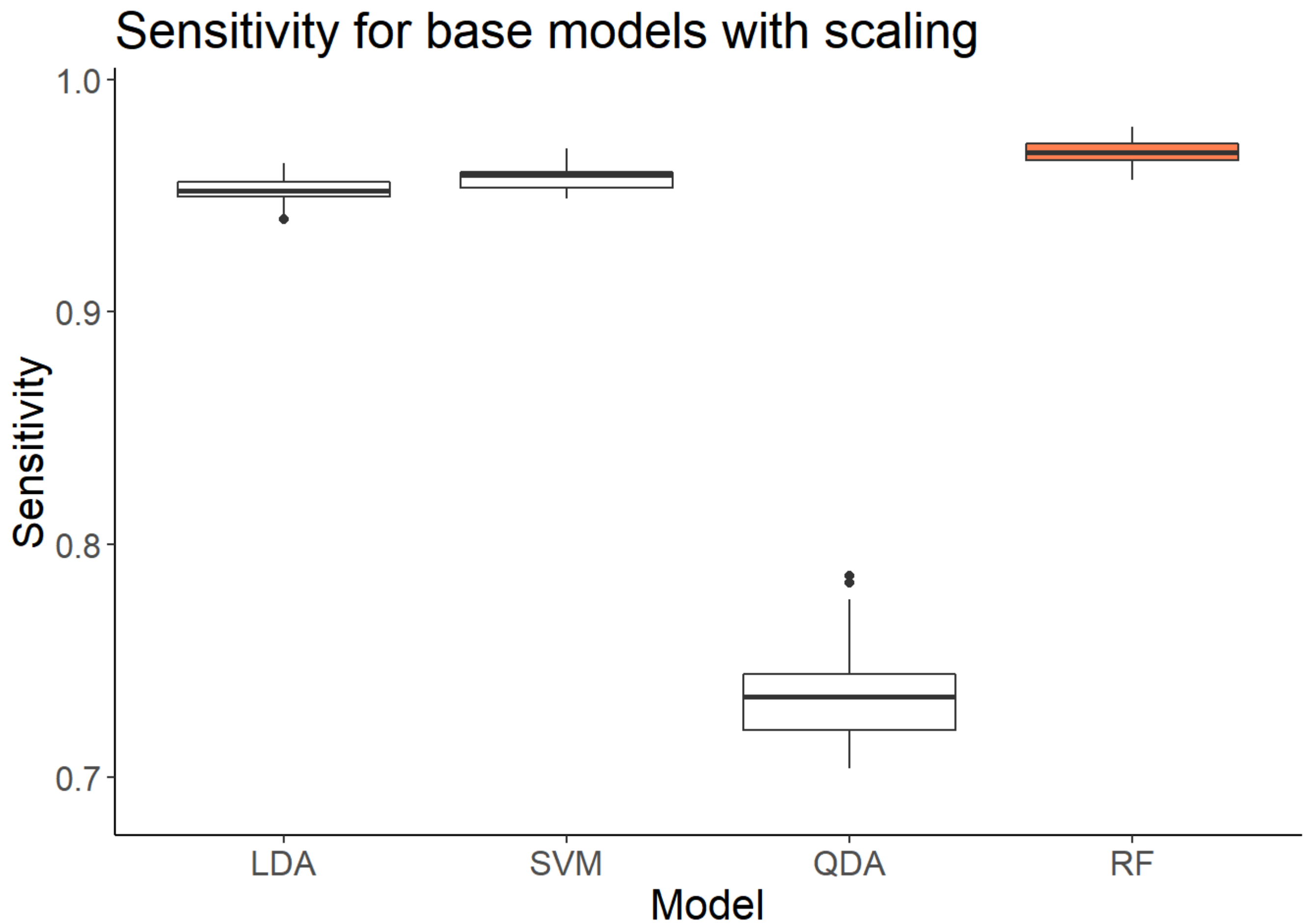
```r
# Rename to avoid conflicts when loading results later
err_mat_melt_scaled <- err_mat_melt
sens_mat_melt_scaled <- sens_mat_melt
spec_mat_melt_scaled <- spec_mat_melt
```

Graph Results

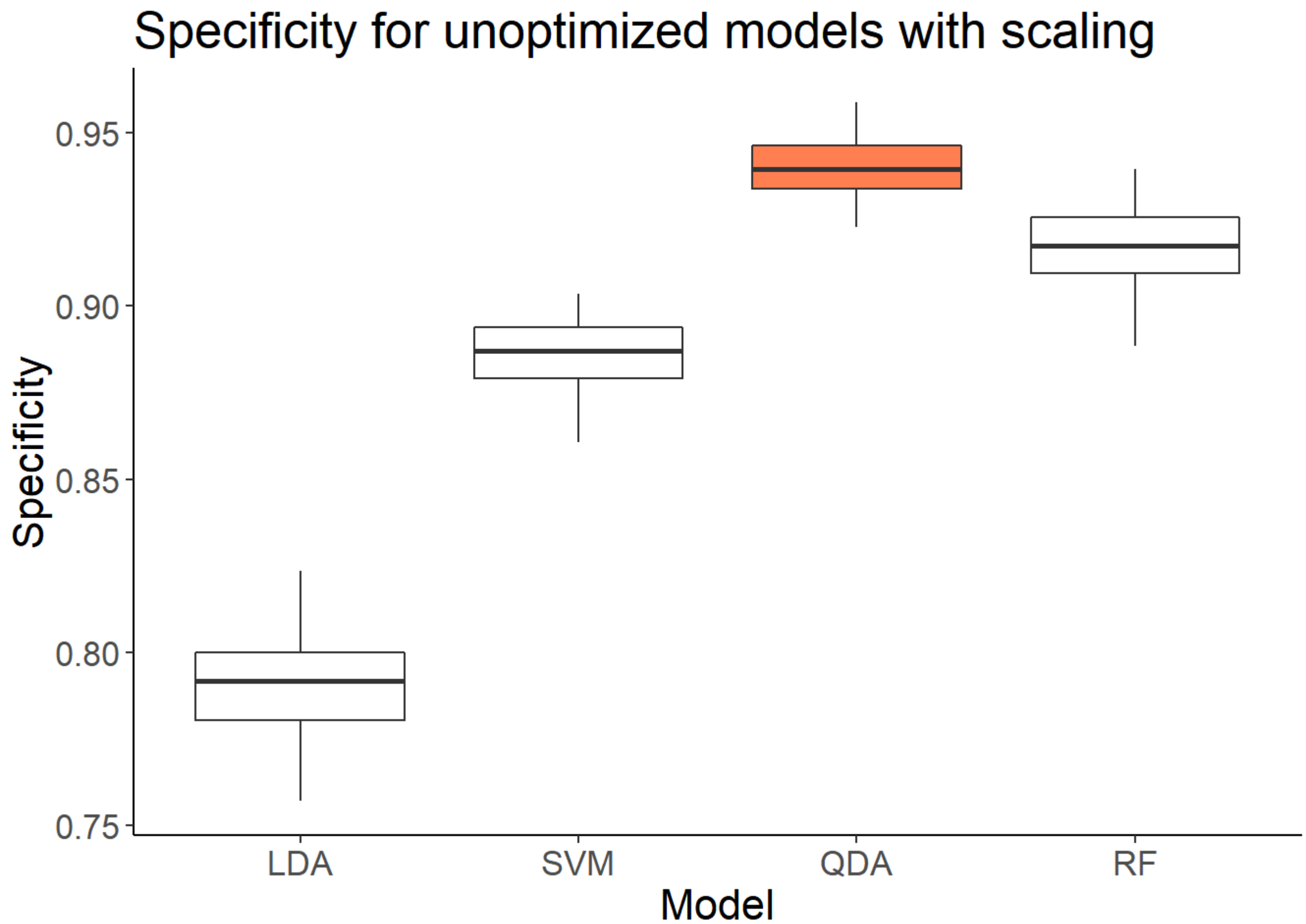
```r
# Plot accuracy for base models
ggplot(err_mat_melt_scaled,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot(fill = c("white", "white", "white", "#FF7F50"))  +
  theme_classic() +
  theme(text = element_text(size = 16),
        axis.title.y = element_blank()) +
  labs(x = "Model", title = "Accuracy for base models with scaling",
       y = "Accuracy") +
  scale_y_continuous(limits = c(0.69, 0.99))
```




```r
# Plot sensitivity for base models
ggplot(sens_mat_melt_scaled,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot(fill = c("white", "white", "white", "#FF7F50"))  +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Model", title = "Sensitivity for base models with scaling") +
  scale_y_continuous(limits = c(0.69, 0.99))
```




```
# Plot specificity for base models
ggplot(spec_mat_melt_scaled,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot(fill = c("white", "white", "#FF7F50", "white"))  +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Model", title = "Specificity for unoptimized models with scaling")
```



Scaling was supposed to help LDA and QDA, but it looks like random forest and SVM are doing better, generally. What if we use the unscaled data?

Try Unscaled Data in Base Models

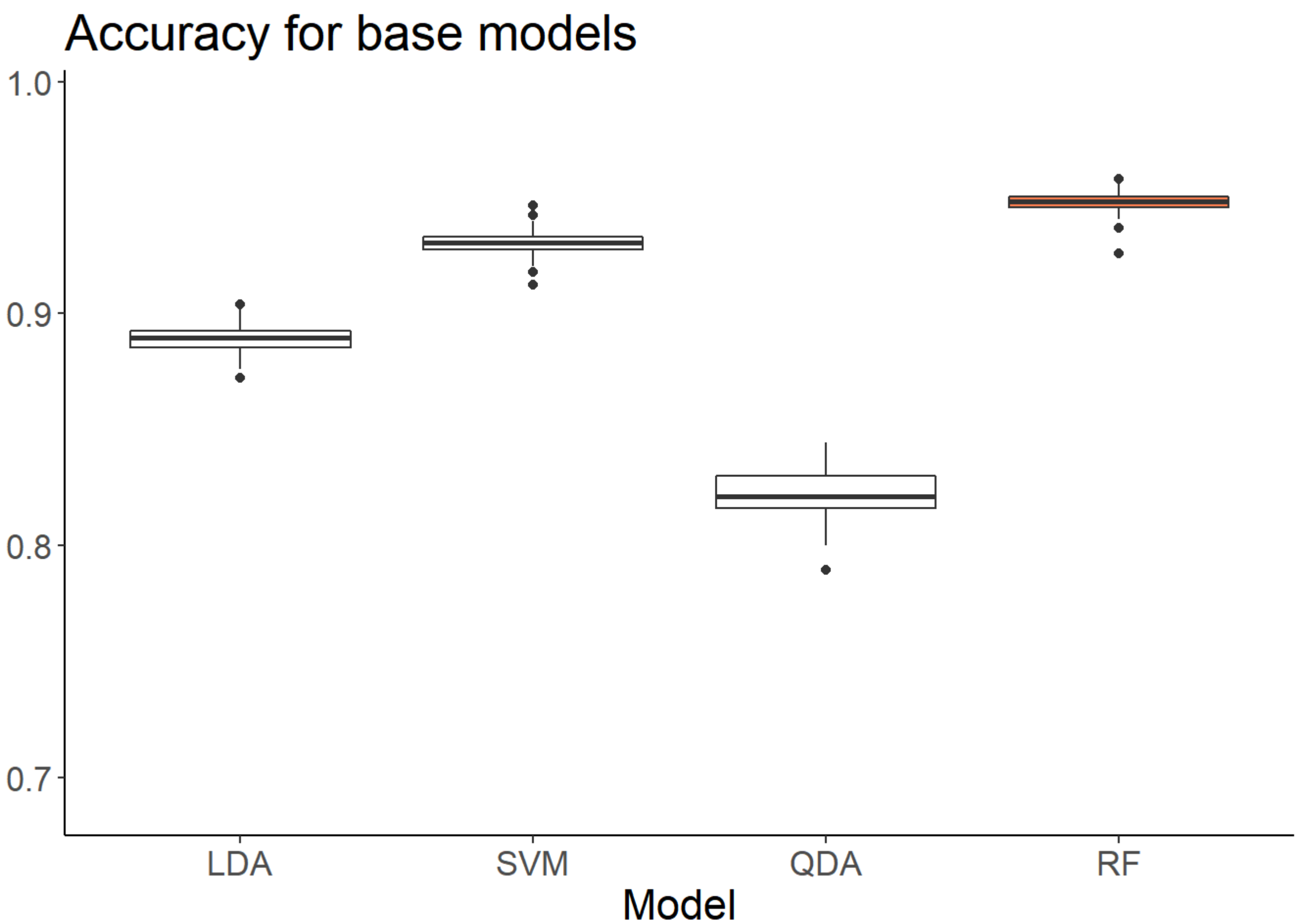
```r
set.seed(1)

# Loop through repetitions
for (r in 1:R){
  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Form training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]
  spam_train_qda = spambase_qda[id$tr,]
  spam_test_qda = spambase_qda[id$ts,]

  # Run random forest model
  mod_rf = randomForest(spam ~ .,
                        spam_train,
                        ntree = 100,
                        trControl = fitControl,
                        metric = "ROC")

  # Run LDA model
  mod_lda <- lda(spam ~ .,
                 spam_train,
                 trControl = fitControl,
                 metric = "ROC")

  # Run QDA model
  mod_qda <- qda(spam ~ .,
                 spam_train_qda,
                 trControl = fitControl,
                 metric = "ROC")

  # Run SVM model
  mod_svm <- svm(spam~.,
                 spam_train,
                 cross=5,
                 type='C-classification',
                 metric = "ROC")

  # Make predictions on test set
  yhat_rf = predict(mod_rf, spam_test[,-58])
  yhat_lda = predict(mod_lda, spam_test[,-58])$class
  yhat_qda = predict(mod_qda, spam_test_qda[,-39])$class
  yhat_svm = predict(mod_svm, spam_test[,-58])

  # Calculate error rates
  err_mat[r,1] = mean(yhat_rf!=spam_test[,58])
  err_mat[r,2] = mean(yhat_lda!=spam_test[,58])
  err_mat[r,3] = mean(yhat_qda!=spam_test_qda[,39])
  err_mat[r,4] = mean(yhat_svm!=spam_test[,58])
```

```
  # Calculate confusion matrices
  cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")
  cm_lda <- confusionMatrix(yhat_lda, spam_test[,58], positive = "No")
  cm_qda <- confusionMatrix(yhat_qda, spam_test_qda[,39], positive = "No")
  cm_svm <- confusionMatrix(yhat_svm, spam_test[,58], positive = "No")

  # Calculate sensitivity
  sens_mat[r,1] = cm_rf$byClass["Sensitivity"]
  sens_mat[r,2] = cm_lda$byClass["Sensitivity"]
  sens_mat[r,3] = cm_qda$byClass["Sensitivity"]
  sens_mat[r,4] = cm_svm$byClass["Sensitivity"]

  # Calculate specificity
  spec_mat[r,1] = cm_rf$byClass["Specificity"]
  spec_mat[r,2] = cm_lda$byClass["Specificity"]
  spec_mat[r,3] = cm_qda$byClass["Specificity"]
  spec_mat[r,4] = cm_svm$byClass["Specificity"]

  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Name output appropriately, melt to prepare for plotting
colnames(err_mat) = c("RF", "LDA", "QDA", "SVM")
err_mat_melt = melt(as.data.frame(err_mat))
colnames(err_mat_melt) = c('Method','Error')

colnames(sens_mat) = c("RF", "LDA", "QDA", "SVM")
sens_mat_melt = melt(as.data.frame(sens_mat))
colnames(sens_mat_melt) = c('Method','Sensitivity')

colnames(spec_mat) = c("RF", "LDA", "QDA", "SVM")
spec_mat_melt = melt(as.data.frame(spec_mat))
colnames(spec_mat_melt) = c('Method','Specificity')
```

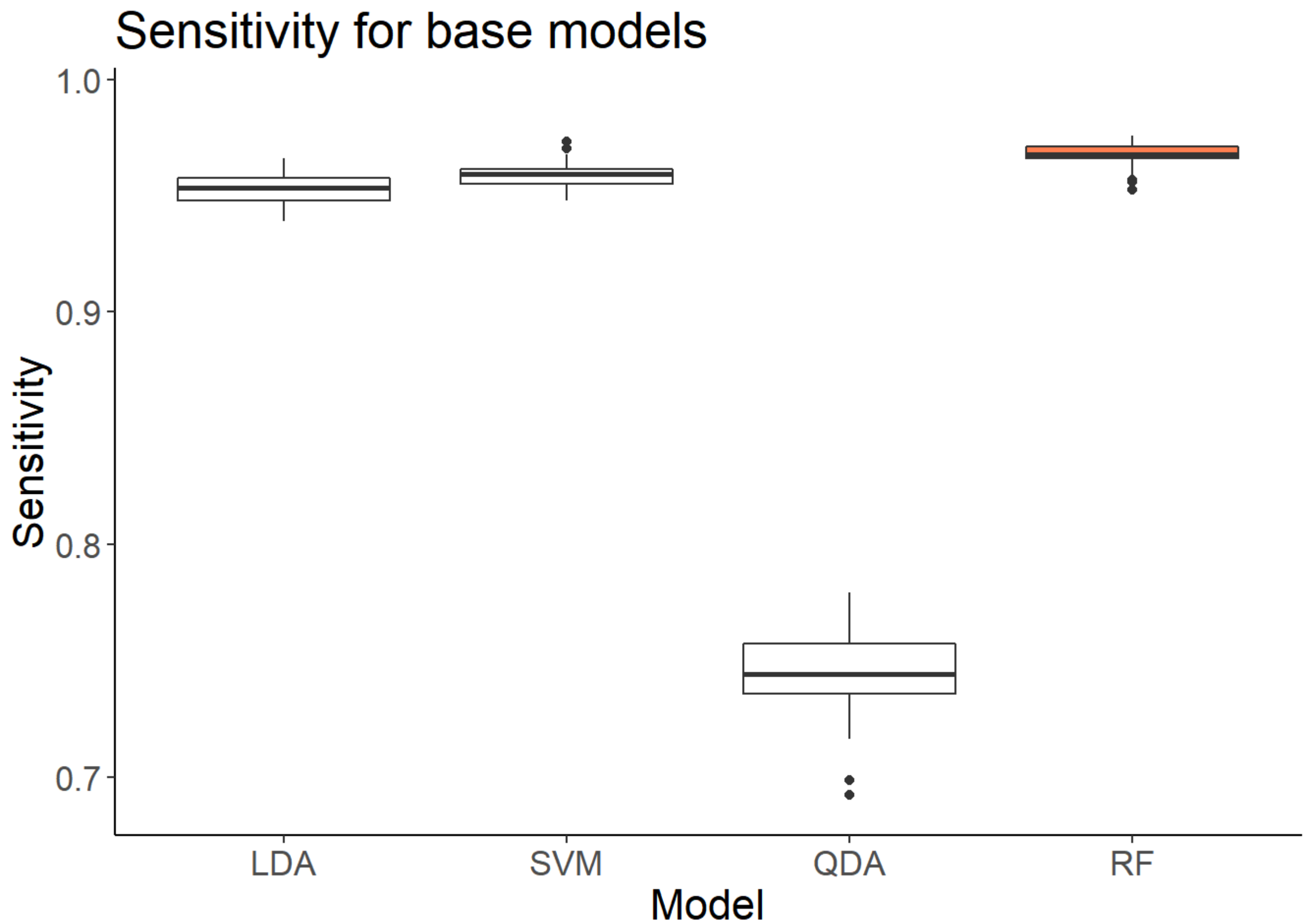
Graph Results

```
# Reorder Method factor levels
err_mat_melt$Method <- ordered(err_mat_melt$Method,
                               levels = c("LDA", "SVM", "QDA", "RF"))

sens_mat_melt$Method <- ordered(sens_mat_melt$Method,
                                levels = c("LDA", "SVM", "QDA", "RF"))

spec_mat_melt$Method <- ordered(spec_mat_melt$Method,
                                levels = c("LDA", "SVM", "QDA", "RF"))
```

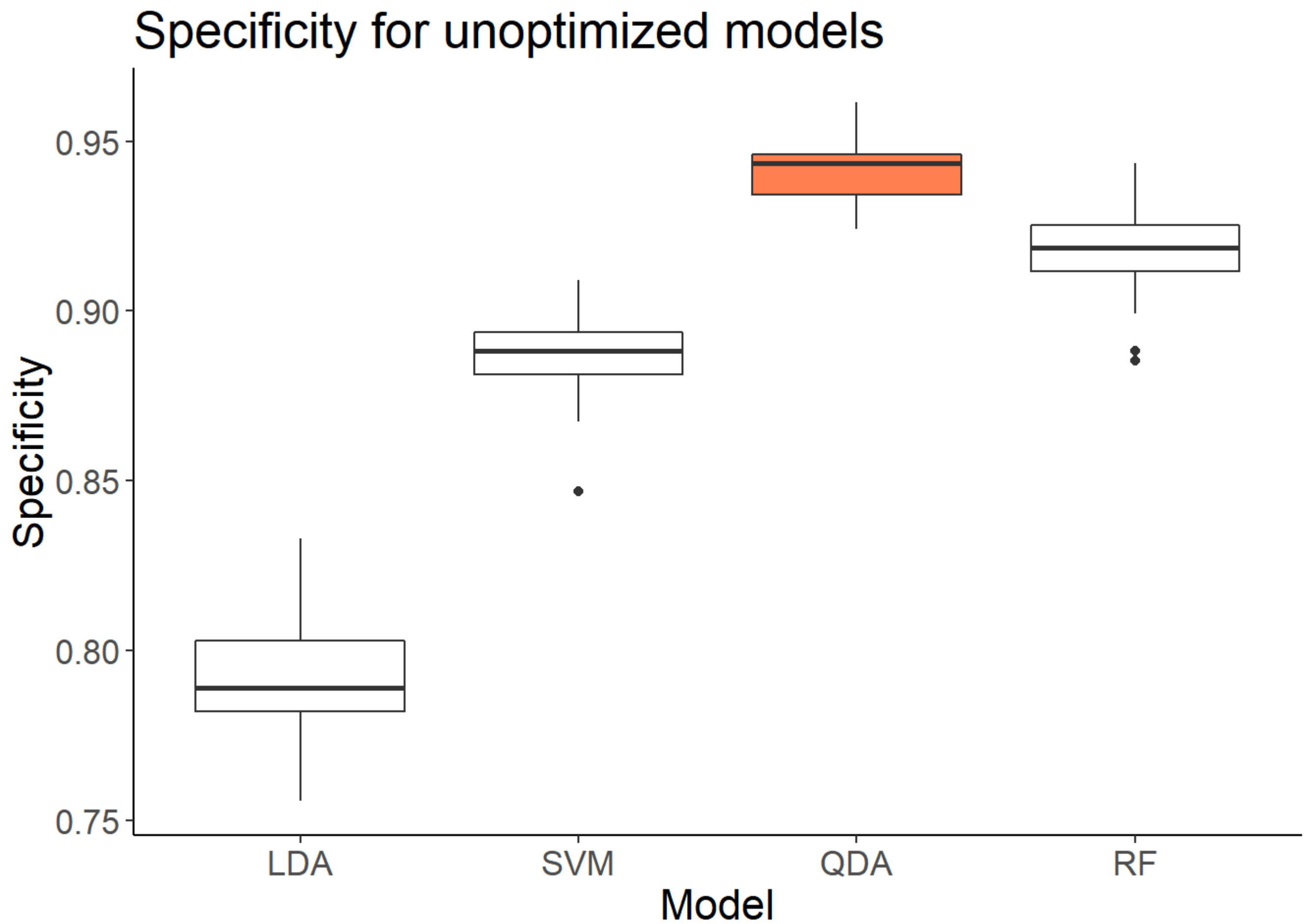
```
# Plot accuracy for base models
ggplot(err_mat_melt,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot(fill = c("white", "white", "white", "#FF7F50"))  +
  theme_classic() +
  theme(text = element_text(size = 16),
        axis.title.y = element_blank()) +
  labs(x = "Model", title = "Accuracy for base models",
       y = "Accuracy") +
  scale_y_continuous(limits = c(0.69, 0.99))
```




```
# Plot sensitivity for base models
ggplot(sens_mat_melt,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot(fill = c("white", "white", "white", "#FF7F50"))  +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Model", title = "Sensitivity for base models") +
  scale_y_continuous(limits = c(0.69, 0.99))
```




```
# Plot specificity for base models
ggplot(spec_mat_melt,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot(fill = c("white", "white", "#FF7F50", "white")) +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Model", title = "Specificity for unoptimized models")
```



Specificity is best in QDA, but that's the least important metric. The results here are very similar, which suggests scaling didn't make a difference. We'll continue with the unscaled version of the data.

Optimization

The random forest model was the best of the unoptimized models. Given the flexibility of SVM, we feel it is also worth trying to optimize.

Check if more trees is better

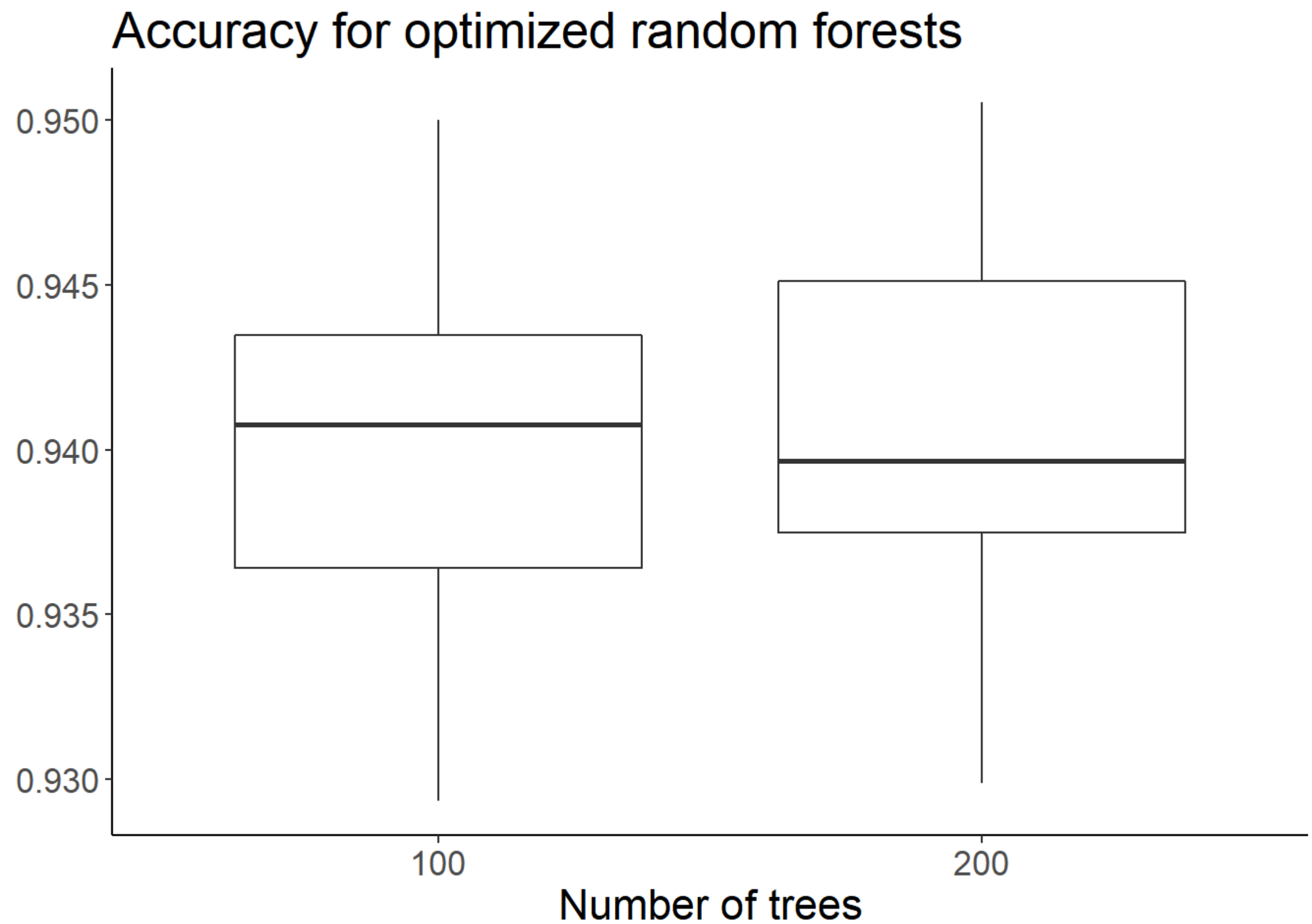
Note: Trying different numbers of trees on top of everything else took too long, so we did an initial run to see if 200 trees was significantly better than 100 (i.e. worth the processing time)

```r
R = 50 # set the number of replications

# set up train control to do CV
fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

set.seed(1)

# Create sequence of values of ntree
num_trees <- c(100, 200)

# create the error matrix to store values
err_mat_opt_rf_trees = matrix(0, ncol=2, nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_rf_trees = matrix(0, ncol=2, nrow=R)

# create specificity matrix to store values
spec_mat_opt_rf_trees = matrix(0, ncol=2, nrow=R)

# Loop through repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through numbers of trees
  for (i in 1:length(num_trees)) {
        mod_rf = train(spam ~ .,
                       spam_train,
                       trControl = fitControl,
                       method = "rf",
                       ntree = num_trees[i],
                       metric = "ROC")

        # Make predictions on test set
        yhat_rf = predict(mod_rf, spam_test[,-58])

        # Calculate error rate
        err_mat_opt_rf_trees[r, i] =
          mean(yhat_rf!=spam_test[,58])

        # Calculate confusion matrix
```

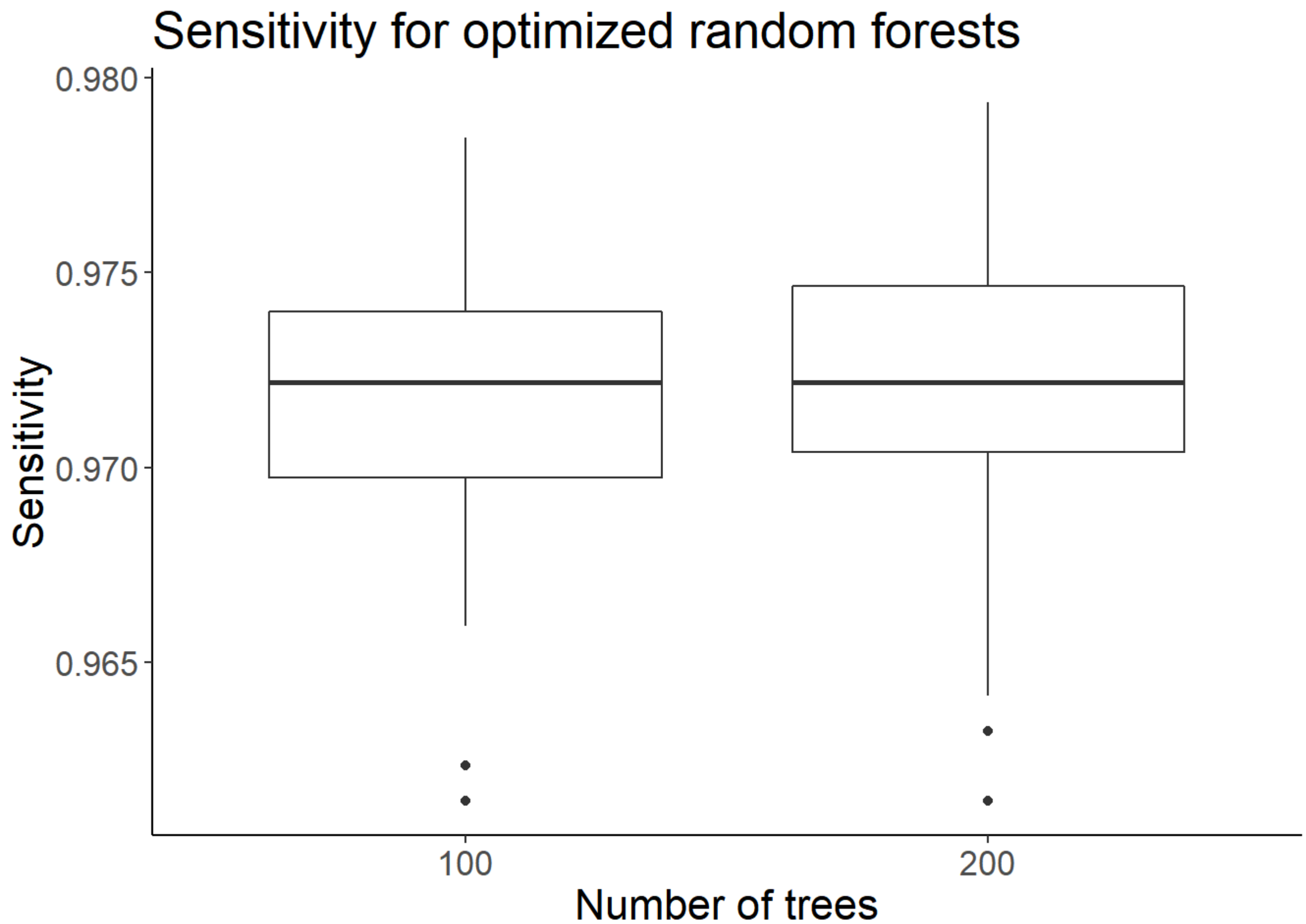
```
        cm_rf_trees <- confusionMatrix(yhat_rf,
                                        spam_test[,58],
                                        positive = "No")

        # Calculate sensitivity
        sens_mat_opt_rf_trees[r, i] =
          cm_rf_trees$byClass["Sensitivity"]

        # Calculate specificity
        spec_mat_opt_rf_trees[r, i] =
          cm_rf_trees$byClass["Specificity"]

  }

    # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_opt_rf_trees = melt(as.data.frame(err_mat_opt_rf_trees))
colnames(err_mat_melt_opt_rf_trees) = c('Method','Error')

sens_mat_melt_opt_rf_trees = melt(as.data.frame(sens_mat_opt_rf_trees))
colnames(sens_mat_melt_opt_rf_trees) = c('Method','Sensitivity')

spec_mat_melt_opt_rf_trees = melt(as.data.frame(spec_mat_opt_rf_trees))
colnames(spec_mat_melt_opt_rf_trees) = c('Method','Specificity')
```

```
err_mat_melt_opt_rf_trees <- err_mat_melt_opt_rf_trees %>%
  mutate(num_trees = as.factor(c(rep(100, 50), rep(200, 50))))
sens_mat_melt_opt_rf_trees <- sens_mat_melt_opt_rf_trees %>%
  mutate(num_trees = as.factor(c(rep(100, 50), rep(200, 50))))
spec_mat_melt_opt_rf_trees <- spec_mat_melt_opt_rf_trees %>%
  mutate(num_trees = as.factor(c(rep(100, 50), rep(200, 50))))
```

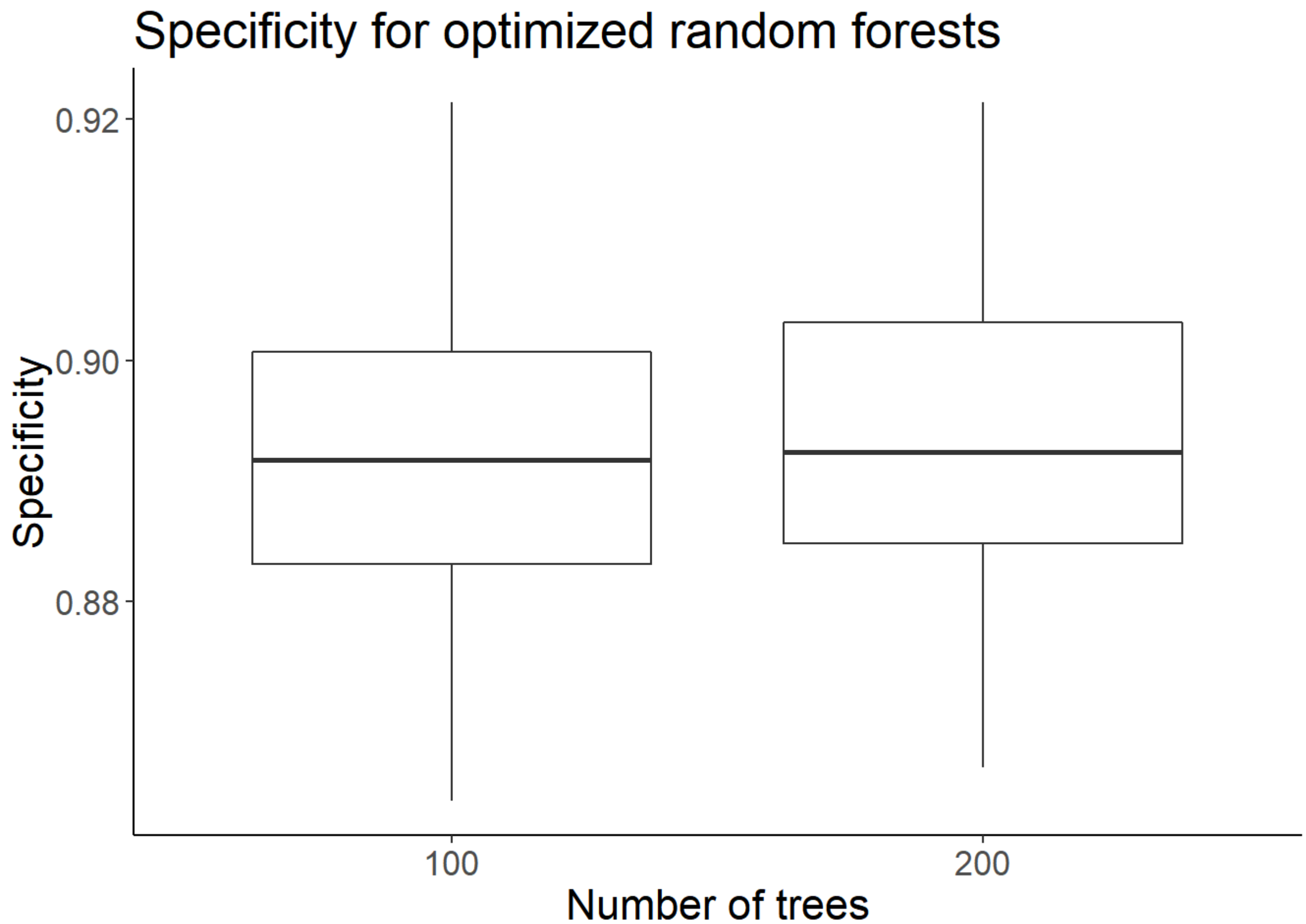
```
ggplot(data = err_mat_melt_opt_rf_trees,
       aes(x = num_trees, y = 1-Error)
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16),
        axis.title.y = element_blank()) +
  labs(x = "Number of trees",
       title = "Accuracy for optimized random forests", y = "Accuracy")
```




```
ggplot(data = sens_mat_melt_opt_rf_trees,
       aes(x = num_trees, y = Sensitivity)
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Number of trees",
       title = "Sensitivity for optimized random forests")
```




```
ggplot(data = spec_mat_melt_opt_rf_trees,
       aes(x = num_trees, y = Specificity)
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Number of trees",
       title = "Specificity for optimized random forests")
```



Spoiler alert: 200 trees isn't much better. We'll stick with 100 for conservation of computing resources.

Try more mtry, weights, c, and nu values

On to the real optimization (with 100 trees)

```
R = 50 # set the number of replications

# set up train control to do CV
fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

set.seed(1)

# Create sequence of values of mtry
num_var <- seq(4, 10, 1)

# Create sequence of values of ntree
num_trees <- c(100)

# Create sequence of weights
weights <- list(c(1, 1), c(1, 0.8), c(1, 0.5))

# create the error matrix to store values
err_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# create specificity matrix to store values
spec_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# Find max allowed value of nu
nu_max <- 2*min(sum(spambase$spam == "Yes"),
                sum(spambase$spam == "No")) / nrow(spambase)

# Set number of values of nu to test
n_nu <- 25

# Set number of values of C to test
n_c <- 25

# Create sequence of values of nu to test
v_nu = seq(0.01,0.78, length=n_nu)

# Create sequence of values of C to test
v_c = seq(2^(-7), 2^7, length=n_c)

# create the error matrix to store values
```

```
err_mat_opt_nu_svm = matrix(0,
                        ncol=n_nu,
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_nu_svm = matrix(0,
                         ncol=n_nu,
                         nrow=R)

# create specificity matrix to store values
spec_mat_opt_nu_svm = matrix(0,
                         ncol=n_nu,
                         nrow=R)

# create the error matrix to store values
err_mat_opt_c_svm = matrix(0,
                        ncol=n_c,
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_c_svm = matrix(0,
                         ncol=n_c,
                         nrow=R)

# create specificity matrix to store values
spec_mat_opt_c_svm = matrix(0,
                         ncol=n_c,
                         nrow=R)

# Loop through repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through values of mtry
  for (i in 1:length(num_var)) {
    # Loop through numbers of trees (not actually used here)
    for (j in 1:length(num_trees)) {
      # Loop through weights
      for (k in 1:length(weights)) {
        # Run random forest model
        mod_rf = train(spam ~ .,
                       spam_train,
                       trControl = fitControl,
                       method = "rf",
                       tuneGrid = expand.grid(mtry = num_var[i]),
                       ntree = num_trees[j],
```

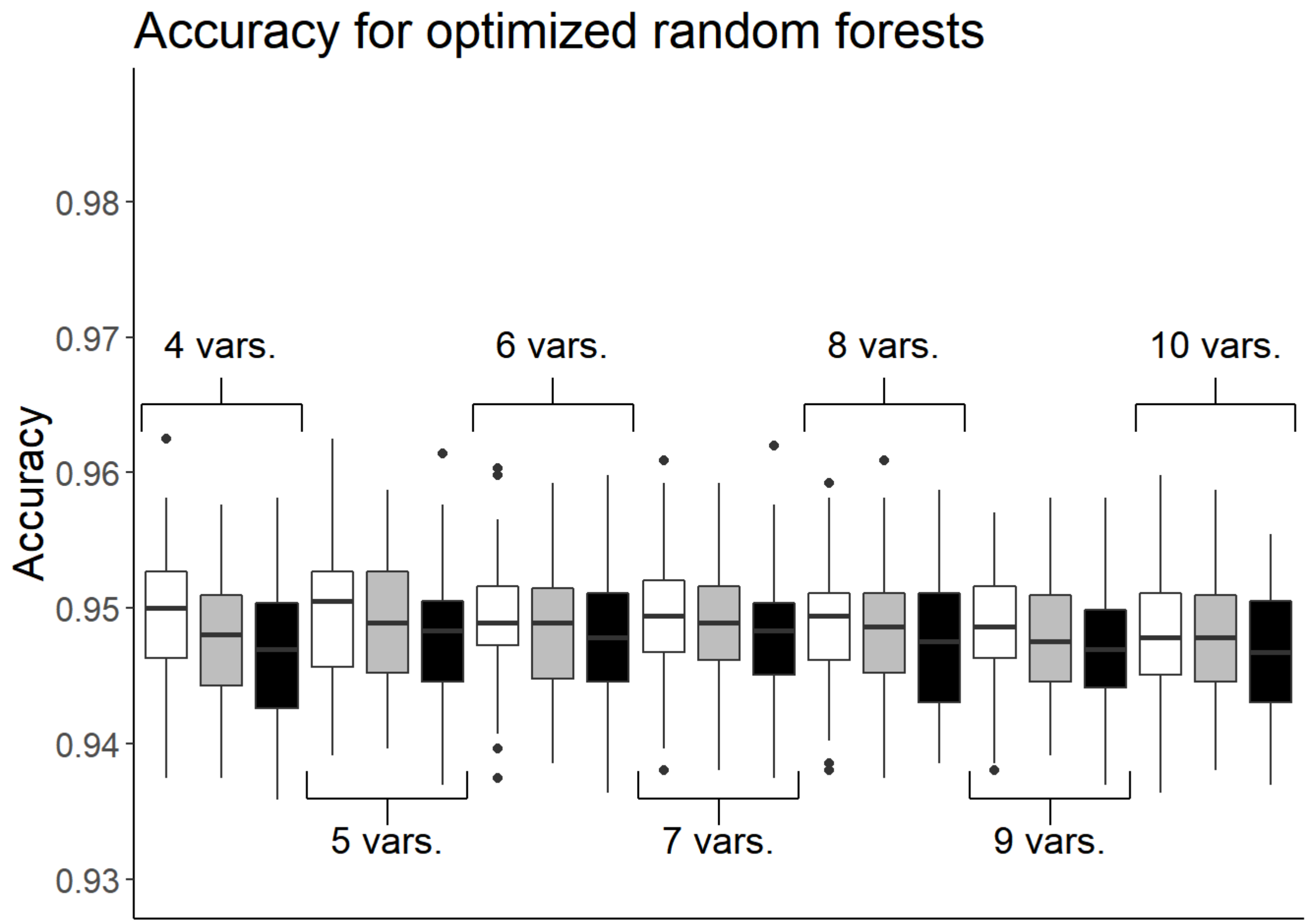
```
                      classwt = weights[[k]],
                      metric = "ROC")

      # Make predictions on test set
      yhat_rf = predict(mod_rf, spam_test[,-58])

      # Calculate the correct column number in which to store output
      index <- (i-1)*length(num_trees)*length(weights) +
        (j-1)*length(weights) + k

      # Calculate error rate
      err_mat_opt_rf[r, index] =
        mean(yhat_rf!=spam_test[,58])

      # Calculate confusion matrix
      cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")

      # Calculate sensitivity
      sens_mat_opt_rf[r, index] =
        cm_rf$byClass["Sensitivity"]

      # Calculate specificity
      spec_mat_opt_rf[r, index] =
        cm_rf$byClass["Specificity"]

    }
  }
}

# Loop through values of nu
for(n in 1:n_nu) {
  # Run nu-SVM model
  mod_nu_svm <- svm(spam~.,
                    spam_train,
                    cross=5,
                    nu=v_nu[n],
                    type='nu-classification',
                    metric = "ROC")

  # Make predictions on test set
  yhat_nu_svm = predict(mod_nu_svm, spam_test[,-58])
  # Calculate error rate
  err_mat_opt_nu_svm[r,n] = mean(yhat_nu_svm!=spam_test[,58])
  # Calculate confusion matrix
  cm_nu_svm <- confusionMatrix(yhat_nu_svm, spam_test[,58],
                               positive = "No")
  # Calculate sensitivity
  sens_mat_opt_nu_svm[r,n] = cm_nu_svm$byClass["Sensitivity"]
  # Calculate specificity
  spec_mat_opt_nu_svm[r,n] = cm_nu_svm$byClass["Specificity"]
}

# Loop through values of C
for(n in 1:n_c) {
```

```
    # Run C-SVM model
    mod_c_svm <- svm(spam~.,
                     spam_train,
                     cross=5,
                     cost=v_c[n],
                     type='C-classification',
                     metric = "ROC")
    # Make predictions on test set
    yhat_c_svm = predict(mod_c_svm, spam_test[,-58])
    # Calculate error rate
    err_mat_opt_c_svm[r,n] = mean(yhat_c_svm!=spam_test[,58])
    # Calculate confusion matrix
    cm_c_svm <- confusionMatrix(yhat_c_svm, spam_test[,58], positive = "No")
    # Calculate sensitivity
    sens_mat_opt_c_svm[r,n] = cm_c_svm$byClass["Sensitivity"]
    # Calculate specificity
    spec_mat_opt_c_svm[r,n] = cm_c_svm$byClass["Specificity"]
  }


  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_opt_rf = melt(as.data.frame(err_mat_opt_rf))
colnames(err_mat_melt_opt_rf) = c('Method','Error')

sens_mat_melt_opt_rf = melt(as.data.frame(sens_mat_opt_rf))
colnames(sens_mat_melt_opt_rf) = c('Method','Sensitivity')

spec_mat_melt_opt_rf = melt(as.data.frame(spec_mat_opt_rf))
colnames(spec_mat_melt_opt_rf) = c('Method','Specificity')

err_mat_melt_opt_nu_svm = melt(as.data.frame(err_mat_opt_nu_svm))
colnames(err_mat_melt_opt_nu_svm) = c('Method','Error')

sens_mat_melt_opt_nu_svm = melt(as.data.frame(sens_mat_opt_nu_svm))
colnames(sens_mat_melt_opt_nu_svm) = c('Method','Sensitivity')

spec_mat_melt_opt_nu_svm = melt(as.data.frame(spec_mat_opt_nu_svm))
colnames(spec_mat_melt_opt_nu_svm) = c('Method','Specificity')

err_mat_melt_opt_c_svm = melt(as.data.frame(err_mat_opt_c_svm))
colnames(err_mat_melt_opt_c_svm) = c('Method','Error')

sens_mat_melt_opt_c_svm = melt(as.data.frame(sens_mat_opt_c_svm))
colnames(sens_mat_melt_opt_c_svm) = c('Method','Sensitivity')

spec_mat_melt_opt_c_svm = melt(as.data.frame(spec_mat_opt_c_svm))
colnames(spec_mat_melt_opt_c_svm) = c('Method','Specificity')
```

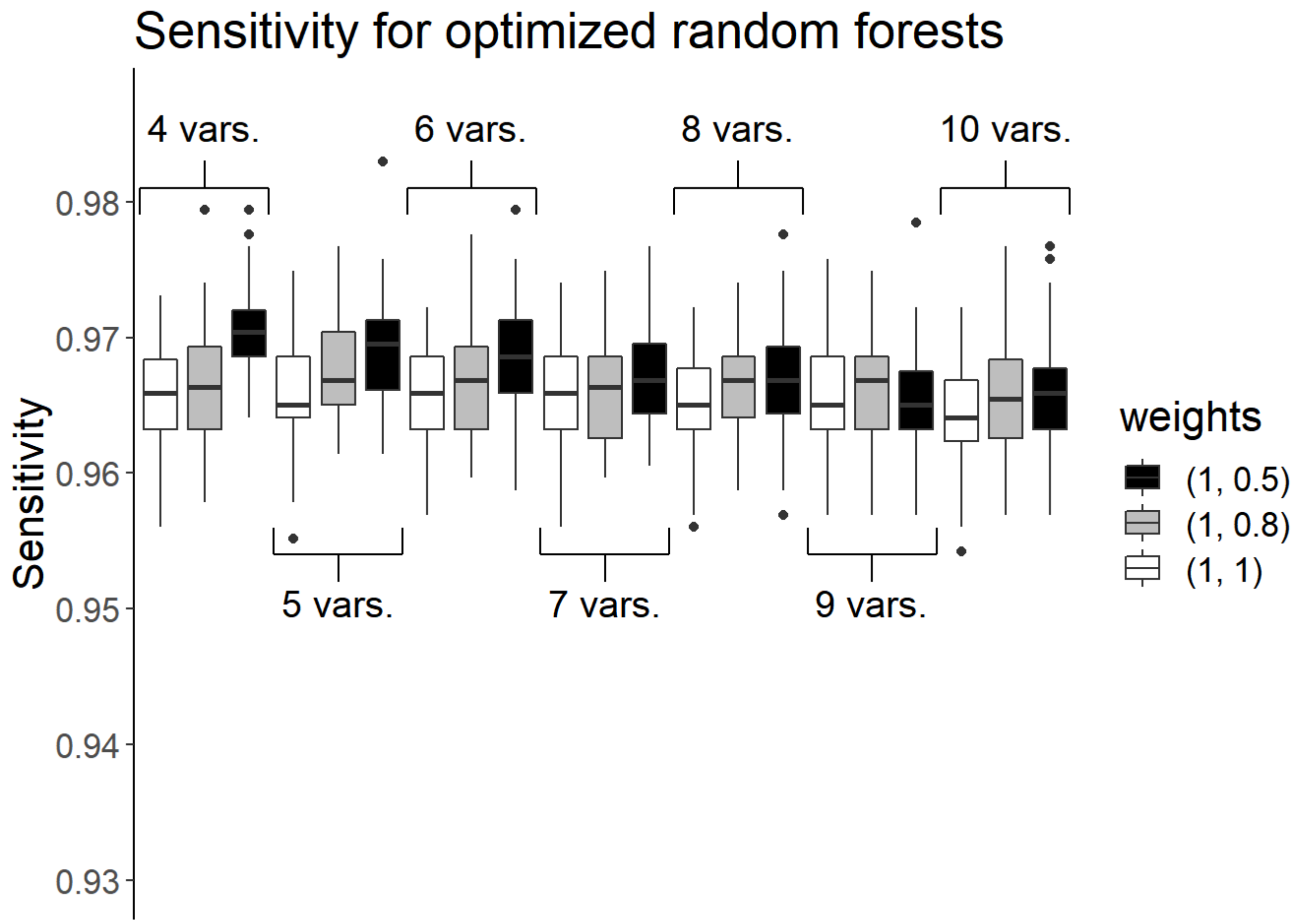
```r
# Label output with descriptions of weights
err_mat_melt_opt_rf <- err_mat_melt_opt_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50), rep("(1, 0.8)", 50),
                         rep("(1, 0.5)", 50)), 7))
sens_mat_melt_opt_rf <- sens_mat_melt_opt_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50), rep("(1, 0.8)", 50),
                         rep("(1, 0.5)", 50)), 7))
spec_mat_melt_opt_rf <- spec_mat_melt_opt_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50), rep("(1, 0.8)", 50),
                         rep("(1, 0.5)", 50)), 7))
```

Graph results

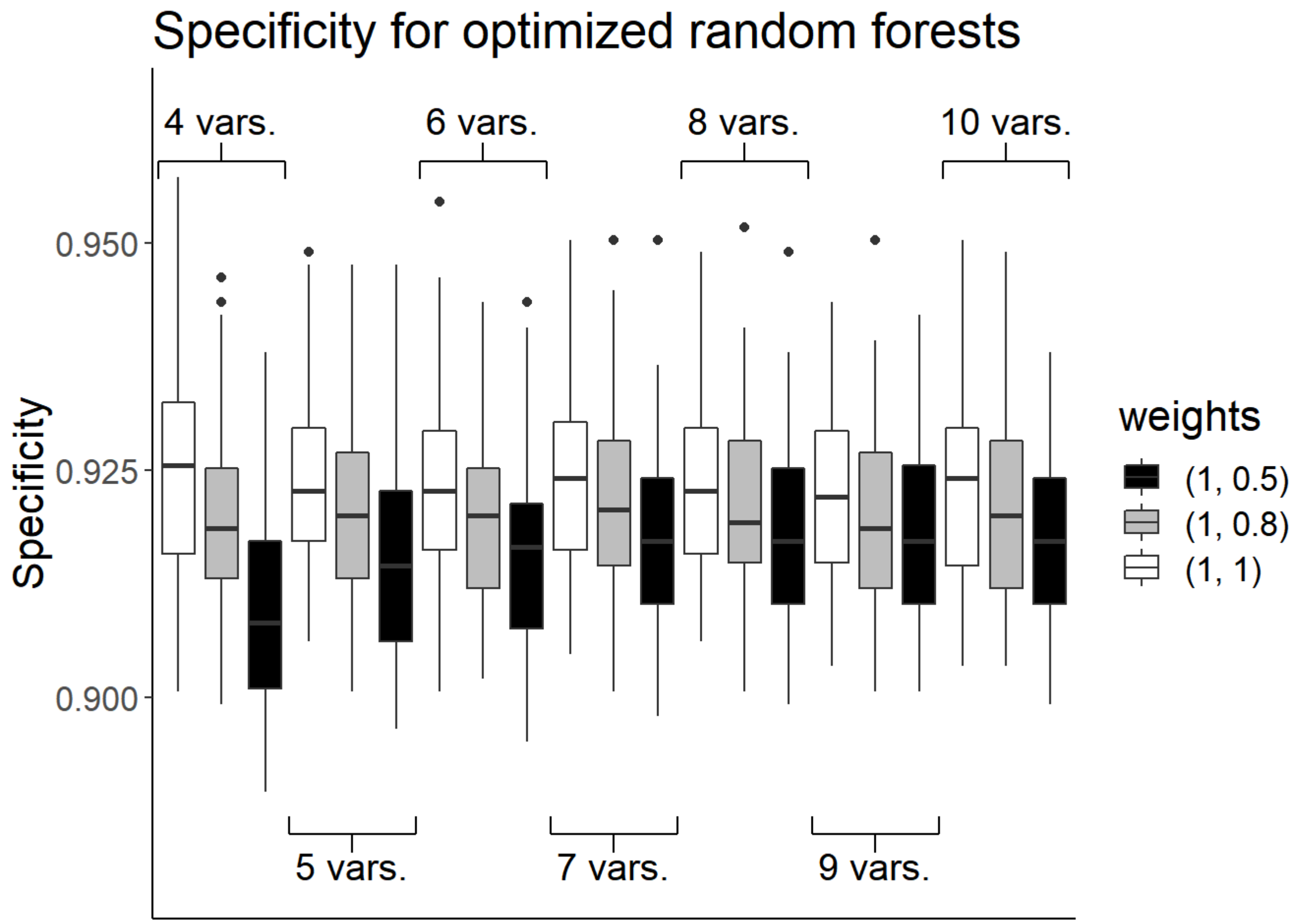
```r
# Plot accuracy for optimized random forests
ggplot(err_mat_melt_opt_rf,mapping=aes(x=Method,y=1-Error,fill=weights))+
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.title.x = element_blank()) +
  labs(title = "Accuracy for optimized random forests", y = "Accuracy") +
  scale_y_continuous(limits = c(0.93, 0.987)) +
  geom_segment(x = 0.55, xend = 0.55, y = 0.963, yend = 0.965) +
  geom_segment(x = 3.45, xend = 3.45, y = 0.963, yend = 0.965) +
  geom_segment(x = 0.55, xend = 3.45, y = 0.965, yend = 0.965) +
  geom_segment(x = 2, xend = 2, y = 0.965, yend = 0.967) +
  annotate("text", label = "4 vars.", x = 2, y = 0.9695, size = 5) +
  geom_segment(x = 3.55, xend = 3.55, y = 0.936, yend = 0.938) +
  geom_segment(x = 6.45, xend = 6.45, y = 0.936, yend = 0.938) +
  geom_segment(x = 3.55, xend = 6.45, y = 0.936, yend = 0.936) +
  geom_segment(x = 5, xend = 5, y = 0.934, yend = 0.936) +
  annotate("text", label = "5 vars.", x = 5, y = 0.933, size = 5) +
  geom_segment(x = 6.55, xend = 6.55, y = 0.963, yend = 0.965) +
  geom_segment(x = 9.45, xend = 9.45, y = 0.963, yend = 0.965) +
  geom_segment(x = 6.55, xend = 9.45, y = 0.965, yend = 0.965) +
  geom_segment(x = 8, xend = 8, y = 0.965, yend = 0.967) +
  annotate("text", label = "6 vars.", x = 8, y = 0.9695, size = 5) +
  geom_segment(x = 9.55, xend = 9.55, y = 0.936, yend = 0.938) +
  geom_segment(x = 12.45, xend = 12.45, y = 0.936, yend = 0.938) +
  geom_segment(x = 9.55, xend = 12.45, y = 0.936, yend = 0.936) +
  geom_segment(x = 11, xend = 11, y = 0.934, yend = 0.936) +
  annotate("text", label = "7 vars.", x = 11, y = 0.933, size = 5) +
  geom_segment(x = 12.55, xend = 12.55, y = 0.963, yend = 0.965) +
  geom_segment(x = 15.45, xend = 15.45, y = 0.963, yend = 0.965) +
  geom_segment(x = 12.55, xend = 15.45, y = 0.965, yend = 0.965) +
  geom_segment(x = 14, xend = 14, y = 0.965, yend = 0.967) +
  annotate("text", label = "8 vars.", x = 14, y = 0.9695, size = 5) +
  geom_segment(x = 15.55, xend = 15.55, y = 0.936, yend = 0.938) +
  geom_segment(x = 18.45, xend = 18.45, y = 0.936, yend = 0.938) +
  geom_segment(x = 15.55, xend = 18.45, y = 0.936, yend = 0.936) +
  geom_segment(x = 17, xend = 17, y = 0.934, yend = 0.936) +
  annotate("text", label = "9 vars.", x = 17, y = 0.933, size = 5) +
  geom_segment(x = 18.55, xend = 18.55, y = 0.963, yend = 0.965) +
  geom_segment(x = 21.45, xend = 21.45, y = 0.963, yend = 0.965) +
  geom_segment(x = 18.55, xend = 21.45, y = 0.965, yend = 0.965) +
  geom_segment(x = 20, xend = 20, y = 0.965, yend = 0.967) +
  annotate("text", label = "10 vars.", x = 20, y = 0.9695, size = 5) +
  scale_fill_manual(values = c("black", "gray", "white"))
```


Accuracy for optimized random forests
Accuracy
0.98
0.97
0.96
0.95
0.94
0.93
4 vars.
5 vars.
6 vars.
7 vars.
8 vars.
9 vars.
10 vars.

```r
# Plot sensitivity for optimized random forests
ggplot(sens_mat_melt_opt_rf,mapping=aes(x=Method,y=Sensitivity,fill=weights))+
  geom_boxplot() +
  theme_classic() +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        text = element_text(size = 16),
        axis.line.x = element_blank()) +
  labs(title = "Sensitivity for optimized random forests") +
  scale_y_continuous(limits = c(0.93, 0.987))  +
  geom_segment(x = 0.55, xend = 0.55, y = 0.979, yend = 0.981) +
  geom_segment(x = 3.45, xend = 3.45, y = 0.979, yend = 0.981) +
  geom_segment(x = 0.55, xend = 3.45, y = 0.981, yend = 0.981)  +
  geom_segment(x = 2, xend = 2, y = 0.981, yend = 0.983) +
  annotate("text", label = "4 vars.", x = 2, y = 0.9855, size = 5) +
  geom_segment(x = 3.55, xend = 3.55, y = 0.956, yend = 0.954) +
  geom_segment(x = 6.45, xend = 6.45, y = 0.956, yend = 0.954) +
  geom_segment(x = 3.55, xend = 6.45, y = 0.954, yend = 0.954) +
  geom_segment(x = 5, xend = 5, y = 0.954, yend = 0.952) +
  annotate("text", label = "5 vars.", x = 5, y = 0.9505, size = 5) +
  geom_segment(x = 6.55, xend = 6.55, y = 0.979, yend = 0.981) +
  geom_segment(x = 9.45, xend = 9.45, y = 0.979, yend = 0.981) +
  geom_segment(x = 6.55, xend = 9.45, y = 0.981, yend = 0.981) +
  geom_segment(x = 8, xend = 8, y = 0.981, yend = 0.983) +
  annotate("text", label = "6 vars.", x = 8, y = 0.9855, size = 5) +
  geom_segment(x = 9.55, xend = 9.55, y = 0.956, yend = 0.954) +
  geom_segment(x = 12.45, xend = 12.45, y = 0.956, yend = 0.954) +
  geom_segment(x = 9.55, xend = 12.45, y = 0.954, yend = 0.954) +
  geom_segment(x = 11, xend = 11, y = 0.954, yend = 0.952) +
  annotate("text", label = "7 vars.", x = 11, y = 0.9505, size = 5) +
  geom_segment(x = 12.55, xend = 12.55, y = 0.979, yend = 0.981) +
  geom_segment(x = 15.45, xend = 15.45, y = 0.979, yend = 0.981) +
  geom_segment(x = 12.55, xend = 15.45, y = 0.981, yend = 0.981) +
  geom_segment(x = 14, xend = 14, y = 0.981, yend = 0.983) +
  annotate("text", label = "8 vars.", x = 14, y = 0.9855, size = 5) +
  geom_segment(x = 15.55, xend = 15.55, y = 0.956, yend = 0.954) +
  geom_segment(x = 18.45, xend = 18.45, y = 0.956, yend = 0.954) +
  geom_segment(x = 15.55, xend = 18.45, y = 0.954, yend = 0.954) +
  geom_segment(x = 17, xend = 17, y = 0.954, yend = 0.952) +
  annotate("text", label = "9 vars.", x = 17, y = 0.9505, size = 5) +
  geom_segment(x = 18.55, xend = 18.55, y = 0.979, yend = 0.981) +
  geom_segment(x = 21.45, xend = 21.45, y = 0.979, yend = 0.981) +
  geom_segment(x = 18.55, xend = 21.45, y = 0.981, yend = 0.981) +
  geom_segment(x = 20, xend = 20, y = 0.981, yend = 0.983) +
  annotate("text", label = "10 vars.", x = 20, y = 0.9855, size = 5) +
  scale_fill_manual(values = c("black", "gray", "white"))
```


Sensitivity for optimized random forests
Sensitivity
0.98
0.97
0.96
0.95
0.94
0.93
4 vars.
5 vars.
6 vars.
7 vars.
8 vars.
9 vars.
10 vars.
weights
(1, 0.5)
(1, 0.8)
(1, 1)

```
# Plot specificity for optimized random forests
ggplot(spec_mat_melt_opt_rf,mapping=aes(x=Method,y=Specificity,fill=weights))+
  geom_boxplot() +
   theme_classic() +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        text = element_text(size = 16)) +
  labs(title = "Specificity for optimized random forests") +
  scale_y_continuous(limits = c(0.88, 0.965)) +
  geom_segment(x = 0.55, xend = 0.55, y = 0.957, yend = 0.959) +
  geom_segment(x = 3.45, xend = 3.45, y = 0.957, yend = 0.959) +
  geom_segment(x = 0.55, xend = 3.45, y = 0.959, yend = 0.959) +
  geom_segment(x = 2, xend = 2, y = 0.959, yend = 0.961) +
  annotate("text", label = "4 vars.", x = 2, y = 0.9635, size = 5) +
  geom_segment(x = 3.55, xend = 3.55, y = 0.887, yend = 0.885) +
  geom_segment(x = 6.45, xend = 6.45, y = 0.887, yend = 0.885) +
  geom_segment(x = 3.55, xend = 6.45, y = 0.885, yend = 0.885) +
  geom_segment(x = 5, xend = 5, y = 0.885, yend = 0.883) +
  annotate("text", label = "5 vars.", x = 5, y = 0.8815, size = 5) +
  geom_segment(x = 6.55, xend = 6.55, y = 0.957, yend = 0.959) +
  geom_segment(x = 9.45, xend = 9.45, y = 0.957, yend = 0.959) +
  geom_segment(x = 6.55, xend = 9.45, y = 0.959, yend = 0.959) +
  geom_segment(x = 8, xend = 8, y = 0.959, yend = 0.961) +
  annotate("text", label = "6 vars.", x = 8, y = 0.9635, size = 5) +
  geom_segment(x = 9.55, xend = 9.55, y = 0.887, yend = 0.885) +
  geom_segment(x = 12.45, xend = 12.45, y = 0.887, yend = 0.885) +
  geom_segment(x = 9.55, xend = 12.45, y = 0.885, yend = 0.885) +
  geom_segment(x = 11, xend = 11, y = 0.885, yend = 0.883) +
  annotate("text", label = "7 vars.", x = 11, y = 0.8815, size = 5) +
  geom_segment(x = 12.55, xend = 12.55, y = 0.957, yend = 0.959) +
  geom_segment(x = 15.45, xend = 15.45, y = 0.957, yend = 0.959) +
  geom_segment(x = 12.55, xend = 15.45, y = 0.959, yend = 0.959) +
  geom_segment(x = 14, xend = 14, y = 0.959, yend = 0.961) +
  annotate("text", label = "8 vars.", x = 14, y = 0.9635, size = 5) +
  geom_segment(x = 15.55, xend = 15.55, y = 0.887, yend = 0.885) +
  geom_segment(x = 18.45, xend = 18.45, y = 0.887, yend = 0.885) +
  geom_segment(x = 15.55, xend = 18.45, y = 0.885, yend = 0.885) +
  geom_segment(x = 17, xend = 17, y = 0.885, yend = 0.883) +
  annotate("text", label = "9 vars.", x = 17, y = 0.8815, size = 5) +
  geom_segment(x = 18.55, xend = 18.55, y = 0.957, yend = 0.959) +
  geom_segment(x = 21.45, xend = 21.45, y = 0.957, yend = 0.959) +
  geom_segment(x = 18.55, xend = 21.45, y = 0.959, yend = 0.959) +
  geom_segment(x = 20, xend = 20, y = 0.959, yend = 0.961) +
  annotate("text", label = "10 vars.", x = 20, y = 0.9635, size = 5) +
  scale_fill_manual(values = c("black", "gray", "white"))
```



Although weighting increases the sensitivity somewhat, it produces unacceptable decreases in specificity. More work needs to be done to determine the optimal number of variables to test at each split. It's also possible that less weighting would produce acceptable declines in specificy. I'm going to try one more round.

Try more weights near the top end

```
R = 50 # set the number of replications

# set up train control to do CV
fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

set.seed(1)

# Create sequence of values of mtry
num_var <- seq(4, 8, 1)

# Create sequence of values of ntree
num_trees <- c(100)

# Create sequence of weights
weights <- list(c(1, 1), c(1, 0.95), c(1, 0.9), c(1,0.85))

# create the error matrix to store values
err_mat_opt2_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt2_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# create specificity matrix to store values
spec_mat_opt2_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# Loop through repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through values of mtry
  for (i in 1:length(num_var)) {
      # Loop through weights
      for (k in 1:length(weights)) {
        # Run random forest model
```

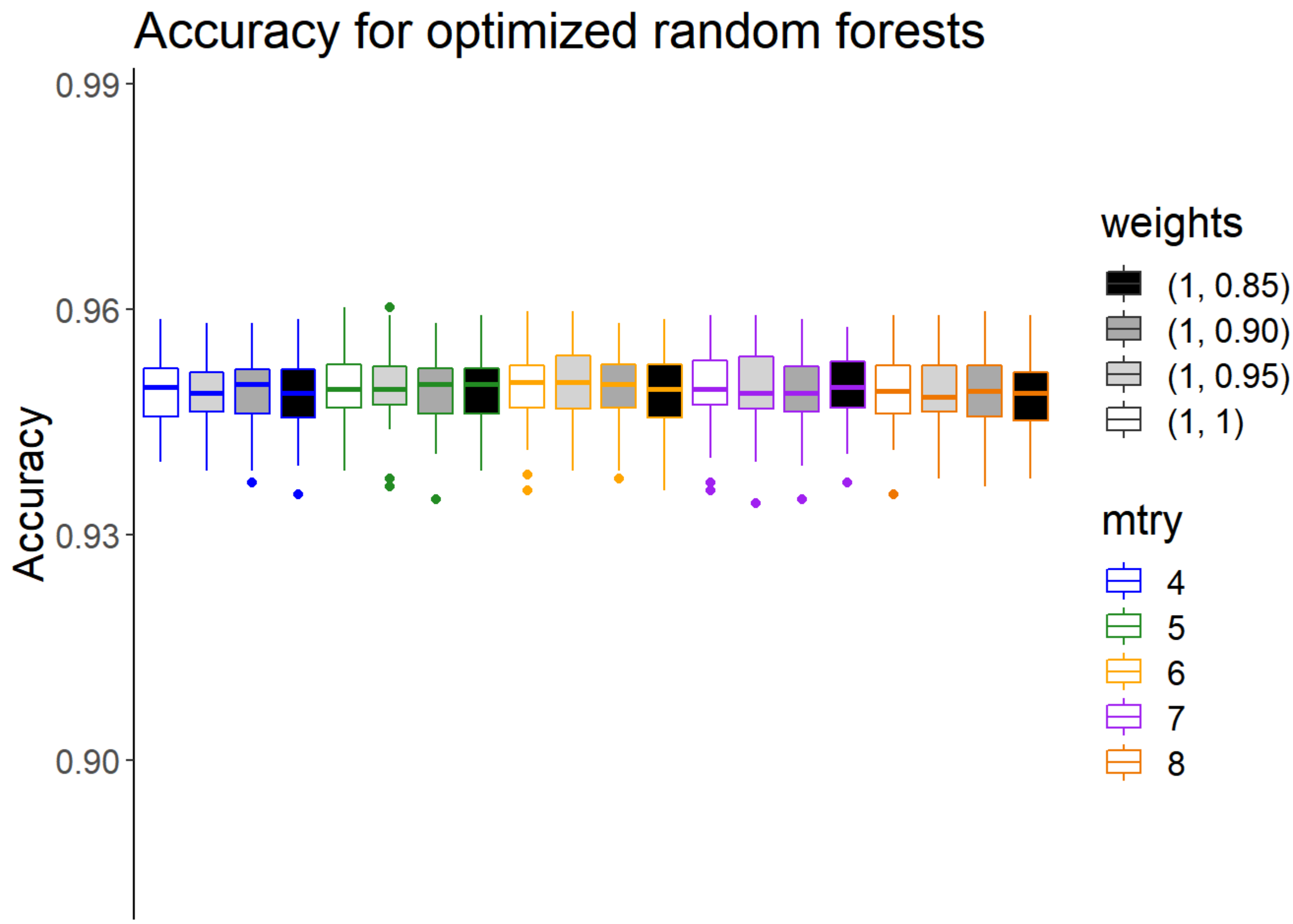
```r
        mod_rf = train(spam ~ .,
                       spam_train,
                       trControl = fitControl,
                       method = "rf",
                       tuneGrid = expand.grid(mtry = num_var[i]),
                       ntree = num_trees,
                       classwt = weights[[k]],
                       metric = "ROC")

        # Make predictions on test set
        yhat_rf = predict(mod_rf, spam_test[,-58])

        # Calculate the correct column number in which to store output
        index <- (i-1)*length(num_trees)*length(weights) +
          (j-1)*length(weights) + k

        # Calculate error rate
        err_mat_opt2_rf[r, index] =
          mean(yhat_rf!=spam_test[,58])

        # Calculate confusion matrix
        cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")

        # Calculate sensitivity
        sens_mat_opt2_rf[r, index] =
          cm_rf$byClass["Sensitivity"]

        # Calculate specificity
        spec_mat_opt2_rf[r, index] =
          cm_rf$byClass["Specificity"]

      }
  }
  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_opt2_rf = melt(as.data.frame(err_mat_opt2_rf))
colnames(err_mat_melt_opt2_rf) = c('Method','Error')

sens_mat_melt_opt2_rf = melt(as.data.frame(sens_mat_opt2_rf))
colnames(sens_mat_melt_opt2_rf) = c('Method','Sensitivity')

spec_mat_melt_opt2_rf = melt(as.data.frame(spec_mat_opt2_rf))
colnames(spec_mat_melt_opt2_rf) = c('Method','Specificity')
```

Graph Results

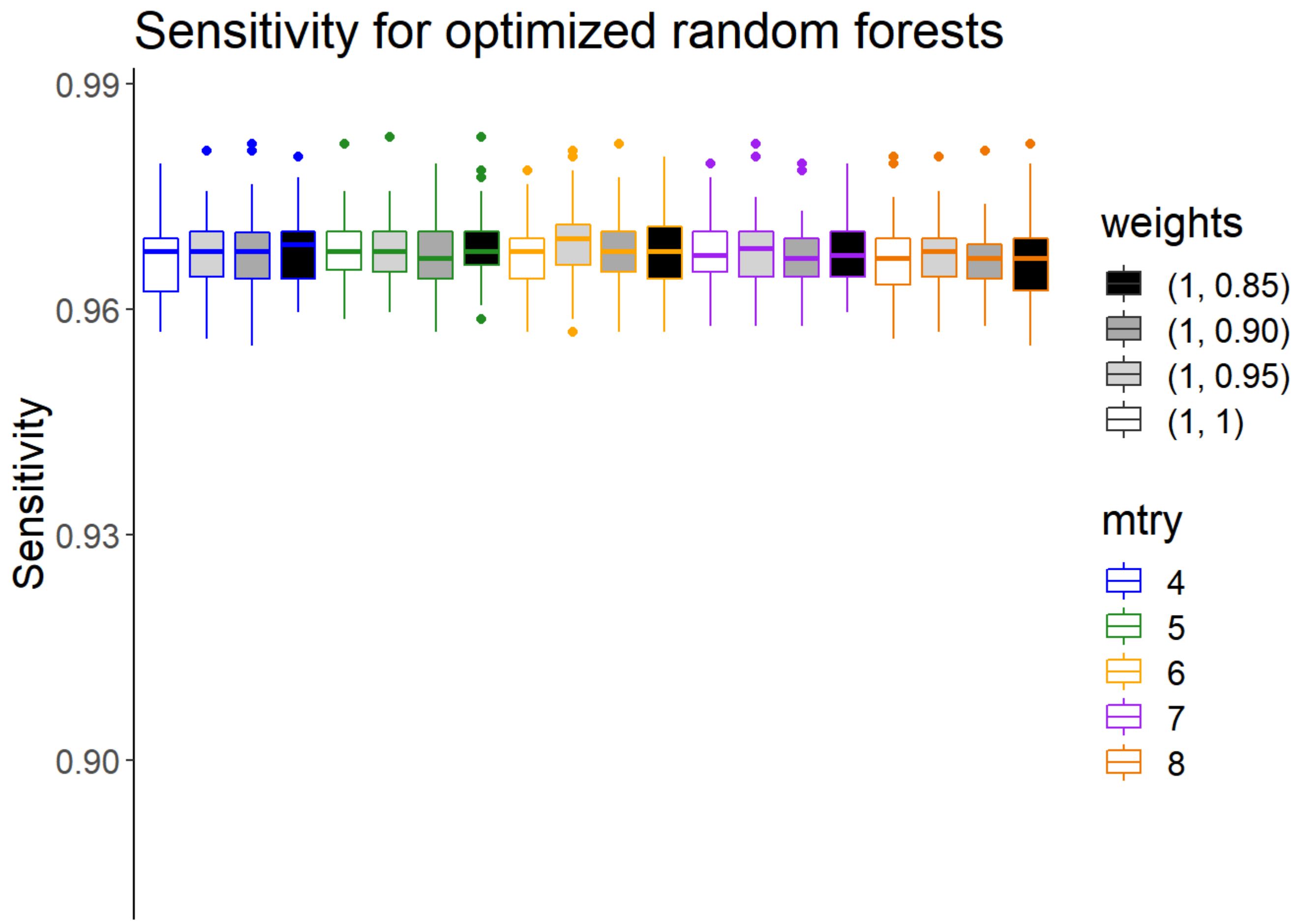
```
# Label output with descriptions of weights and mtry
err_mat_melt_opt2_rf <- err_mat_melt_opt2_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50),
                        rep("(1, 0.95)", 50),
                        rep("(1, 0.90)", 50),
                        rep("(1, 0.85)", 50)),
                      5),
         mtry = as.character(c(rep(4, 200),
                  rep(5, 200),
                  rep(6, 200),
                  rep(7, 200),
                  rep(8, 200))))

sens_mat_melt_opt2_rf <- sens_mat_melt_opt2_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50),
                        rep("(1, 0.95)", 50),
                        rep("(1, 0.90)", 50),
                        rep("(1, 0.85)", 50)),
                      5),
         mtry = as.character(c(rep(4, 200),
                  rep(5, 200),
                  rep(6, 200),
                  rep(7, 200),
                  rep(8, 200))))

spec_mat_melt_opt2_rf <- spec_mat_melt_opt2_rf %>%
  mutate(weights = rep(c(rep("(1, 1)", 50),
                        rep("(1, 0.95)", 50),
                        rep("(1, 0.90)", 50),
                        rep("(1, 0.85)", 50)),
                      5),
         mtry = as.character(c(rep(4, 200),
                  rep(5, 200),
                  rep(6, 200),
                  rep(7, 200),
                  rep(8, 200))))
```

Produce boxplots

```
ggplot(err_mat_melt_opt2_rf,mapping=aes(x=Method,y=1-Error,fill=weights, color = mtry))+
  geom_boxplot() +
  theme_classic() +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        text = element_text(size = 16),
        axis.line.x = element_blank()) +
  labs(title = "Accuracy for optimized random forests") +
  ylab("Accuracy") +
  scale_y_continuous(limits = c(0.884, 0.987))  +
  scale_fill_manual(values = c("black", "darkgrey", "lightgrey", "white")) +
  scale_color_manual(values = c("blue", "forestgreen", "orange", "purple", "darkorange2"))
```

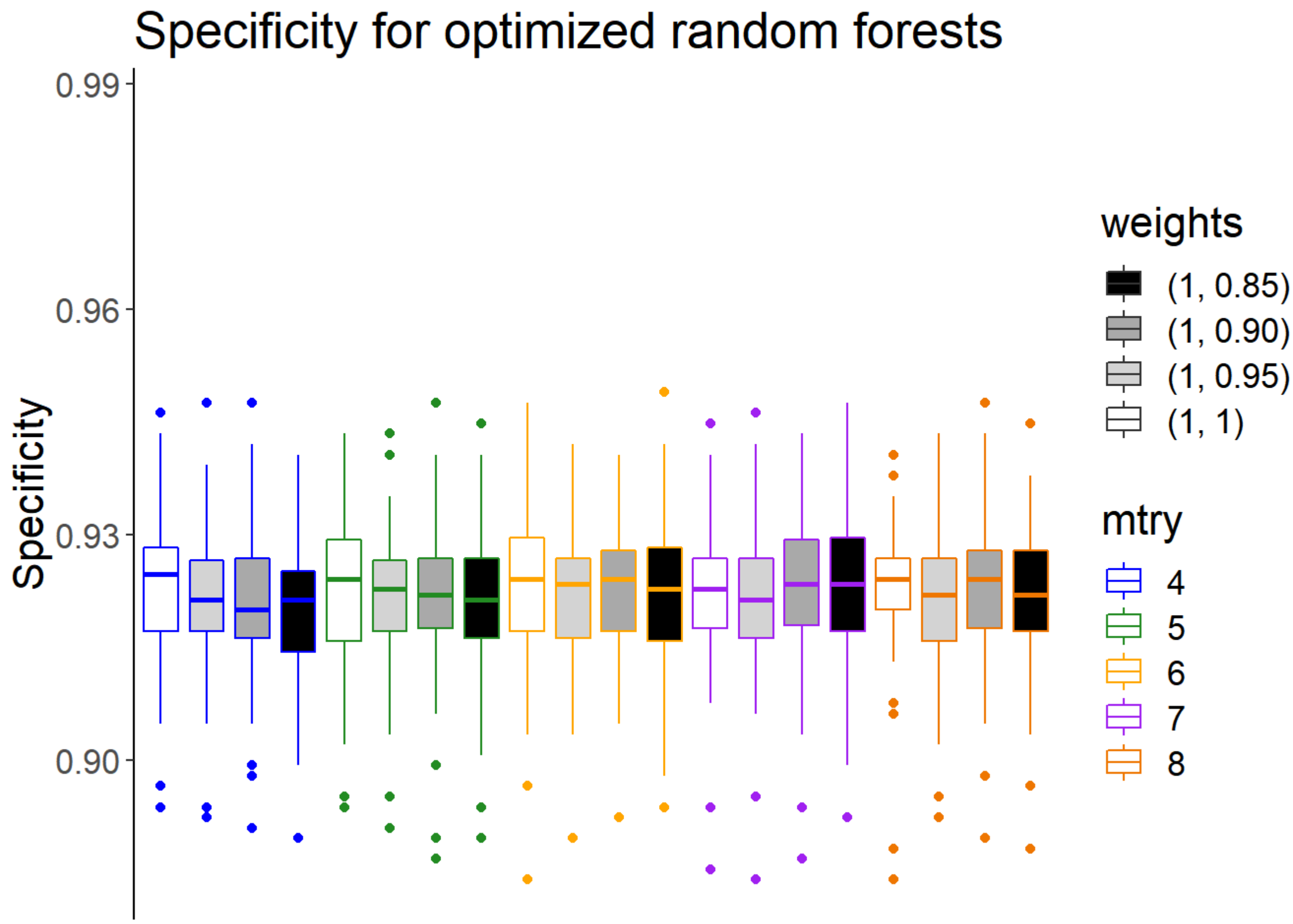



```
ggplot(sens_mat_melt_opt2_rf,mapping=aes(x=Method,y=Sensitivity,fill=weights, color = mtry))+
  geom_boxplot() +
  theme_classic() +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        text = element_text(size = 16),
        axis.line.x = element_blank()) +
  labs(title = "Sensitivity for optimized random forests") +
  scale_y_continuous(limits = c(0.884, 0.987))  +
  scale_fill_manual(values = c("black", "darkgrey", "lightgrey", "white")) +
  scale_color_manual(values = c("blue", "forestgreen", "orange", "purple", "darkorange2"))
```




```
ggplot(spec_mat_melt_opt2_rf,mapping=aes(x=Method, y=Specificity, fill=weights, color = mtry))+
  geom_boxplot() +
  theme_classic() +
  theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        text = element_text(size = 16),
        axis.line.x = element_blank()) +
  labs(title = "Specificity for optimized random forests") +
  scale_y_continuous(limits = c(0.884, 0.987))  +
  scale_fill_manual(values = c("black", "darkgrey", "lightgrey", "white")) +
  scale_color_manual(values = c("blue", "forestgreen", "orange", "purple", "darkorange2"))
```

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



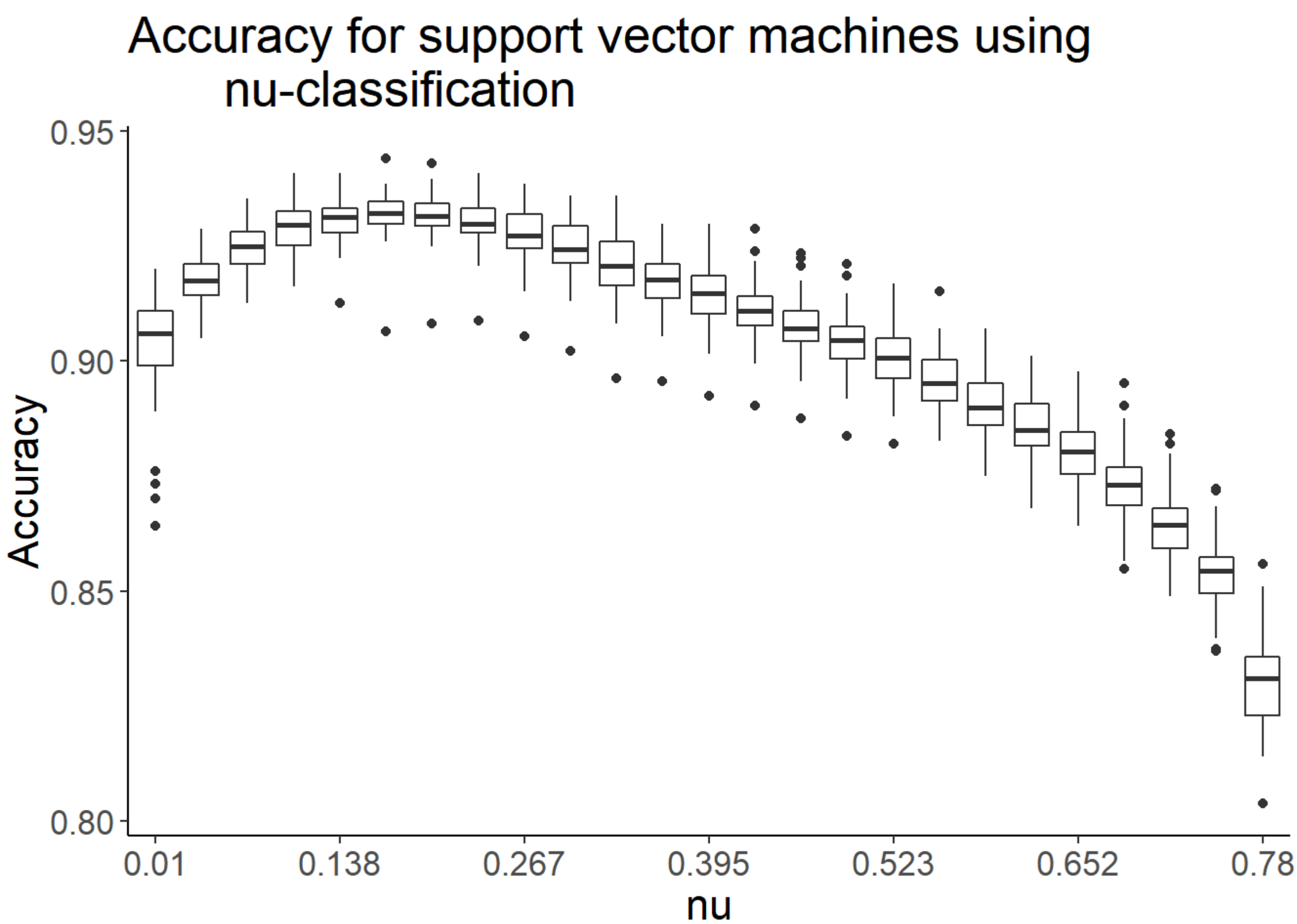
Accuracy medians are very even across the board. The improvement of slight weighting at mtry of 6 and 7 on specificity is counterweighted by its worse performance on specificity (more or less) — the accuracy results appears to indicate that on the particular run when sensitivity is good, specificity is bad, keeping the same overall accuracy (more or less). The weighting may simply encourage overfitting. Therefore, it looks like on the whole weighting doesn't do much good and might not be worth the extra processing time.

Back to evaluation of previous nu-SVM and c-SVM results

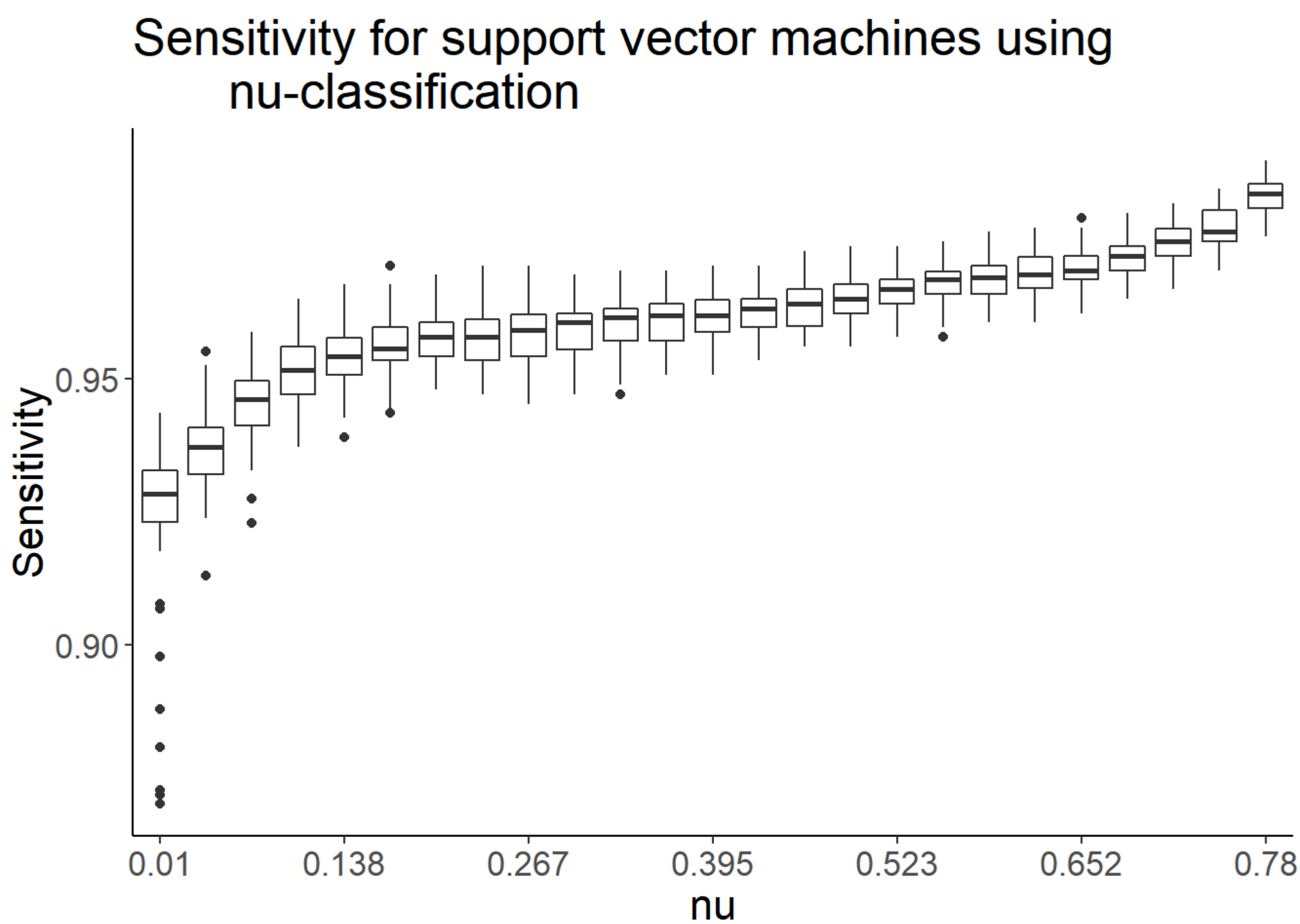
```
# Set number of values of nu tested
n_nu <- 25

# Set number of values of C tested
n_c <- 25

# Create sequence of values of nu tested
v_nu = seq(0.01,0.78, length=n_nu)

# Create sequence of values of C tested
v_c = seq(2^(-7), 2^7, length=n_c)
```

```
# Add column of nu values to aid in plotting
err_mat_melt_opt_nu_svm <- err_mat_melt_opt_nu_svm %>%
  mutate(nu_value = as.factor(sort(rep(v_nu, 50))))
sens_mat_melt_opt_nu_svm <- sens_mat_melt_opt_nu_svm %>%
  mutate(nu_value = as.factor(sort(rep(v_nu, 50))))
spec_mat_melt_opt_nu_svm <- spec_mat_melt_opt_nu_svm %>%
  mutate(nu_value = as.factor(sort(rep(v_nu, 50))))
```

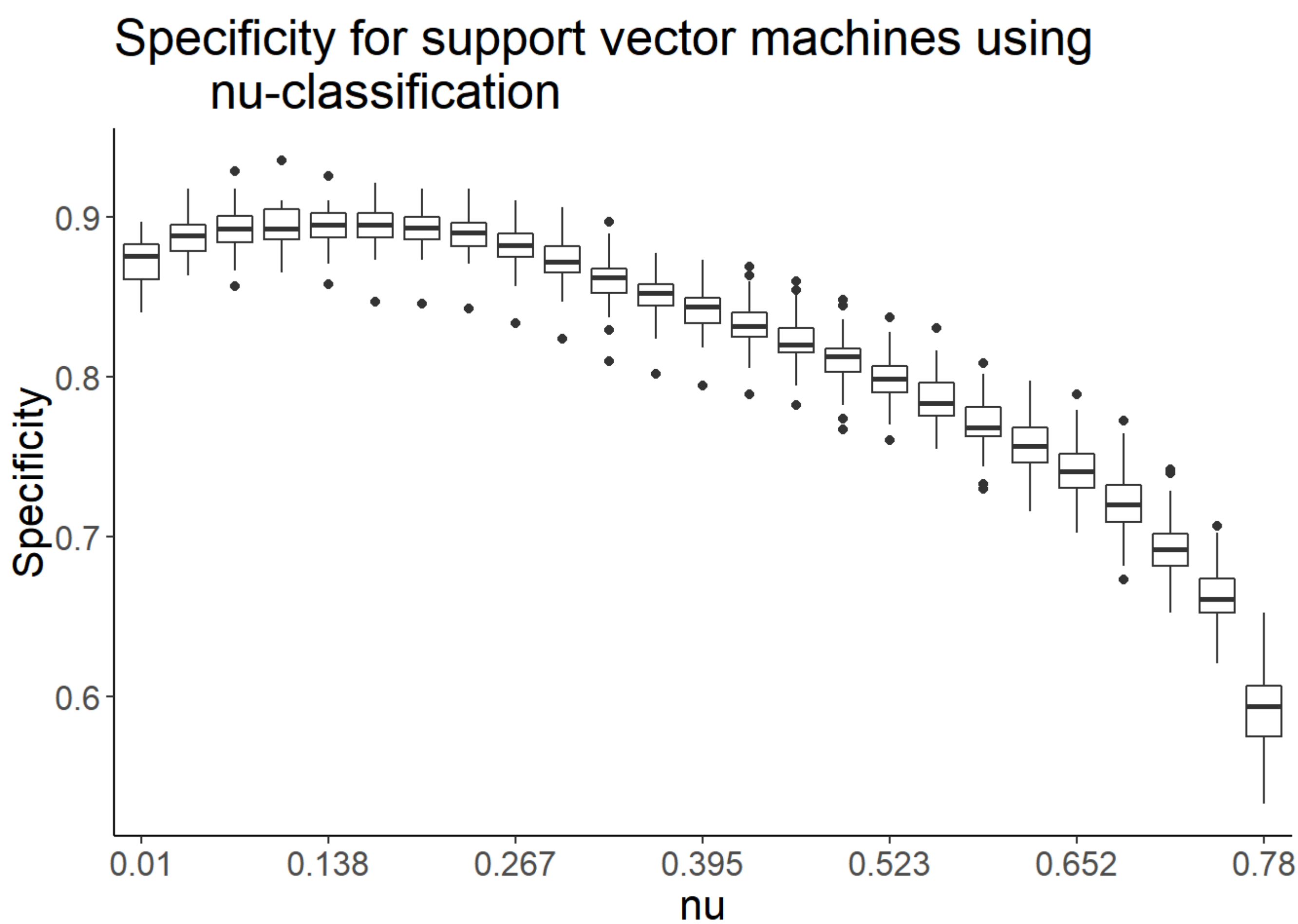
```
# Plot accuracy for nu-SVM
ggplot(data = err_mat_melt_opt_nu_svm,
       aes(x = nu_value, y = 1-Error)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_nu[1], v_nu[5], v_nu[9], v_nu[13], v_nu[17],
                              v_nu[21], v_nu[25]),
                   labels = c(round(v_nu[1], 3), round(v_nu[5], 3),
                              round(v_nu[9], 3), round(v_nu[13], 3),
                              round(v_nu[17], 3), round(v_nu[21], 3),
                              round(v_nu[25], 3))) +
  labs(x = "nu", title = "Accuracy for support vector machines using
       nu-classification", y = "Accuracy")
```



```r
# Plot sensitivity for nu-SVM
ggplot(data = sens_mat_melt_opt_nu_svm,
       aes(x = nu_value, y = Sensitivity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_nu[1], v_nu[5], v_nu[9], v_nu[13], v_nu[17],
                              v_nu[21], v_nu[25]),
                   labels = c(round(v_nu[1], 3), round(v_nu[5], 3),
                              round(v_nu[9], 3), round(v_nu[13], 3),
                              round(v_nu[17], 3), round(v_nu[21], 3),
                              round(v_nu[25], 3))) +
  labs(x = "nu", title = "Sensitivity for support vector machines using
       nu-classification")
```



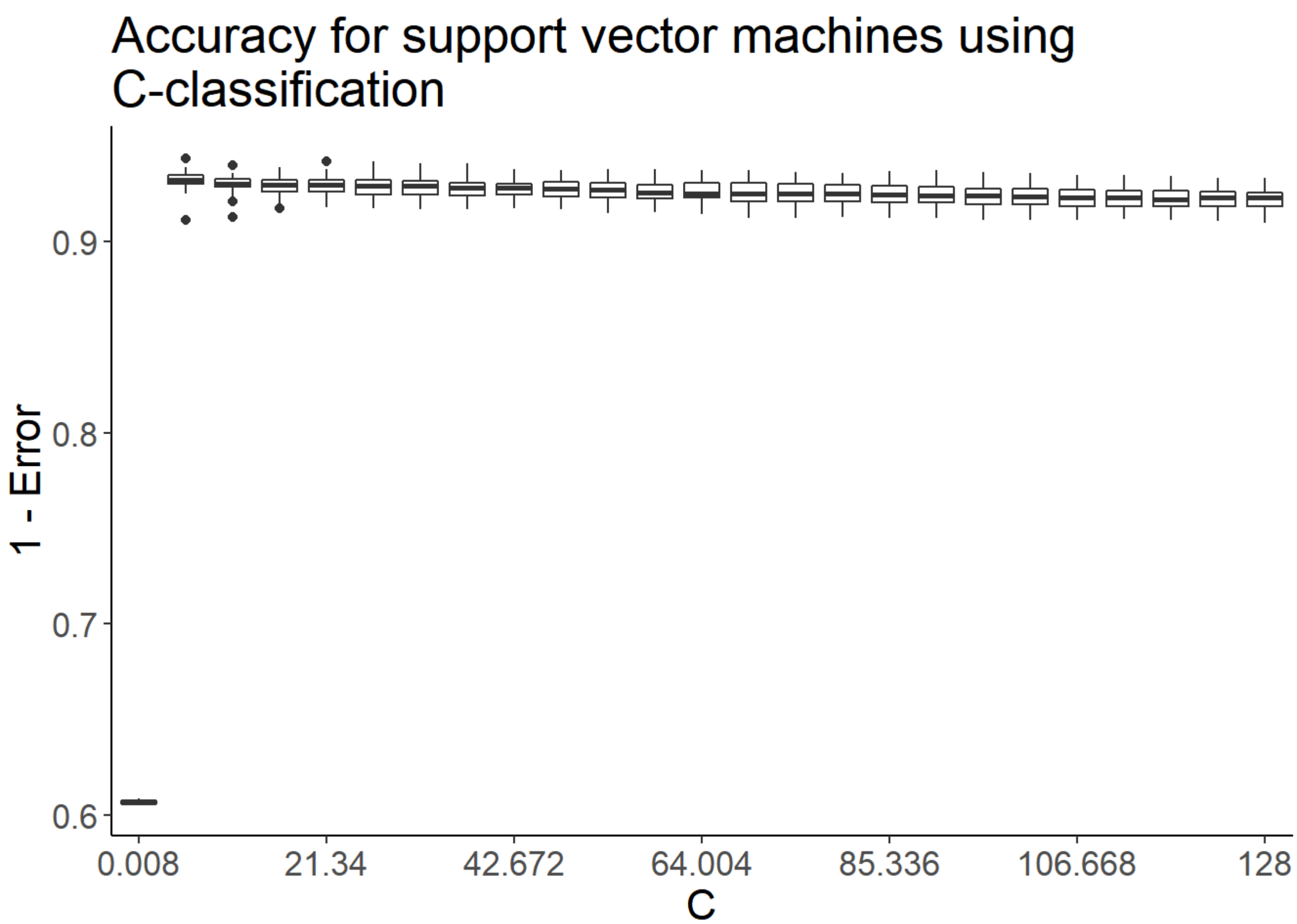
```
# Plot specificity for nu-SVM
ggplot(data = spec_mat_melt_opt_nu_svm,
       aes(x = nu_value, y = Specificity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_nu[1], v_nu[5], v_nu[9], v_nu[13], v_nu[17],
                              v_nu[21], v_nu[25]),
                   labels = c(round(v_nu[1], 3), round(v_nu[5], 3),
                              round(v_nu[9], 3), round(v_nu[13], 3),
                              round(v_nu[17], 3), round(v_nu[21], 3),
                              round(v_nu[25], 3))) +
  labs(x = "nu", title = "Specificity for support vector machines using
       nu-classification")
```



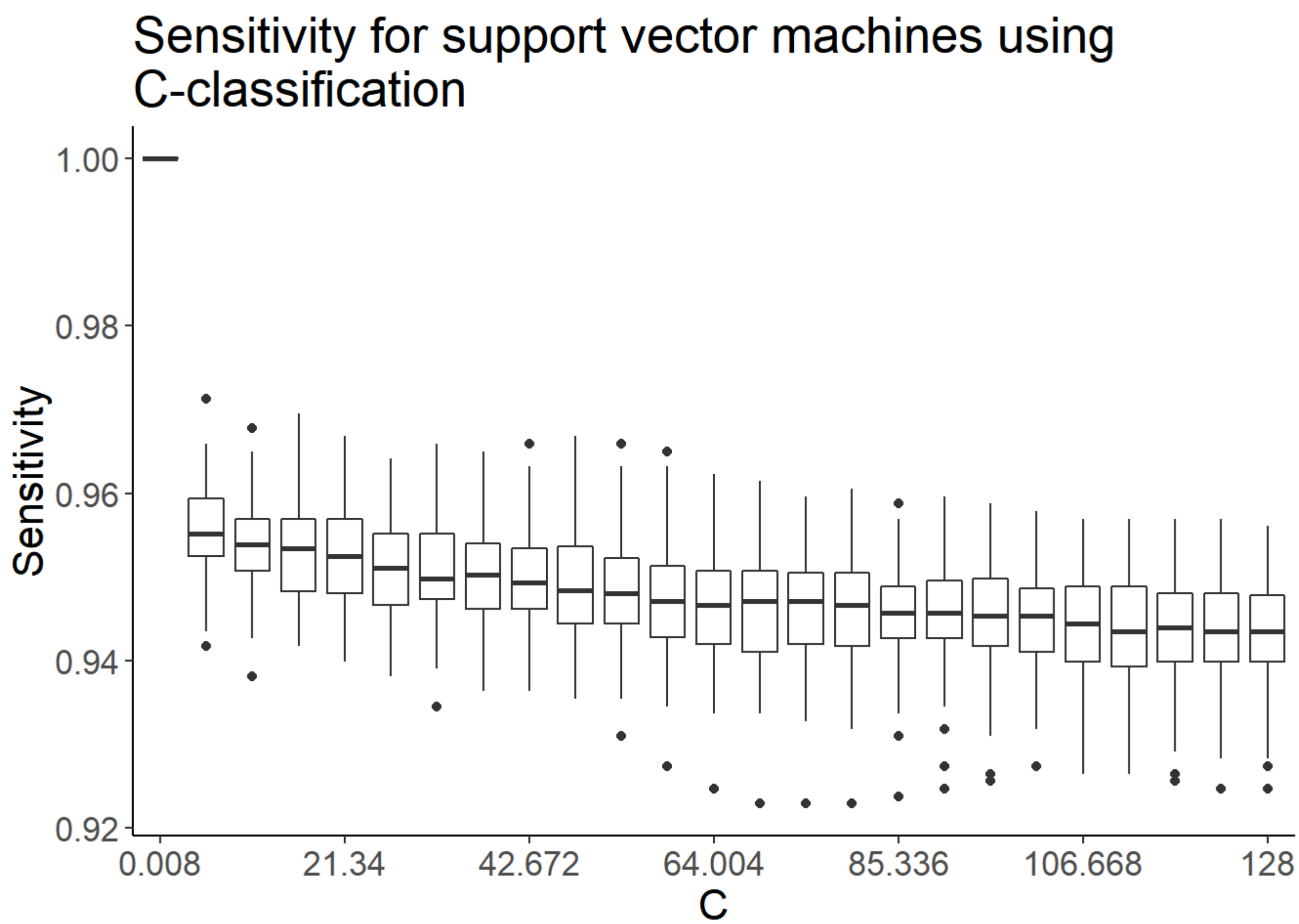
There is no value of nu for which nu-SVM performs as well as the optimized random forests.

```
# Add column of C values to aid in plotting
err_mat_melt_opt_c_svm <- err_mat_melt_opt_c_svm %>%
  mutate(c_value = as.factor(sort(rep(v_c, 50))))
sens_mat_melt_opt_c_svm <- sens_mat_melt_opt_c_svm %>%
  mutate(c_value = as.factor(sort(rep(v_c, 50))))
spec_mat_melt_opt_c_svm <- spec_mat_melt_opt_c_svm %>%
  mutate(c_value = as.factor(sort(rep(v_c, 50))))
```

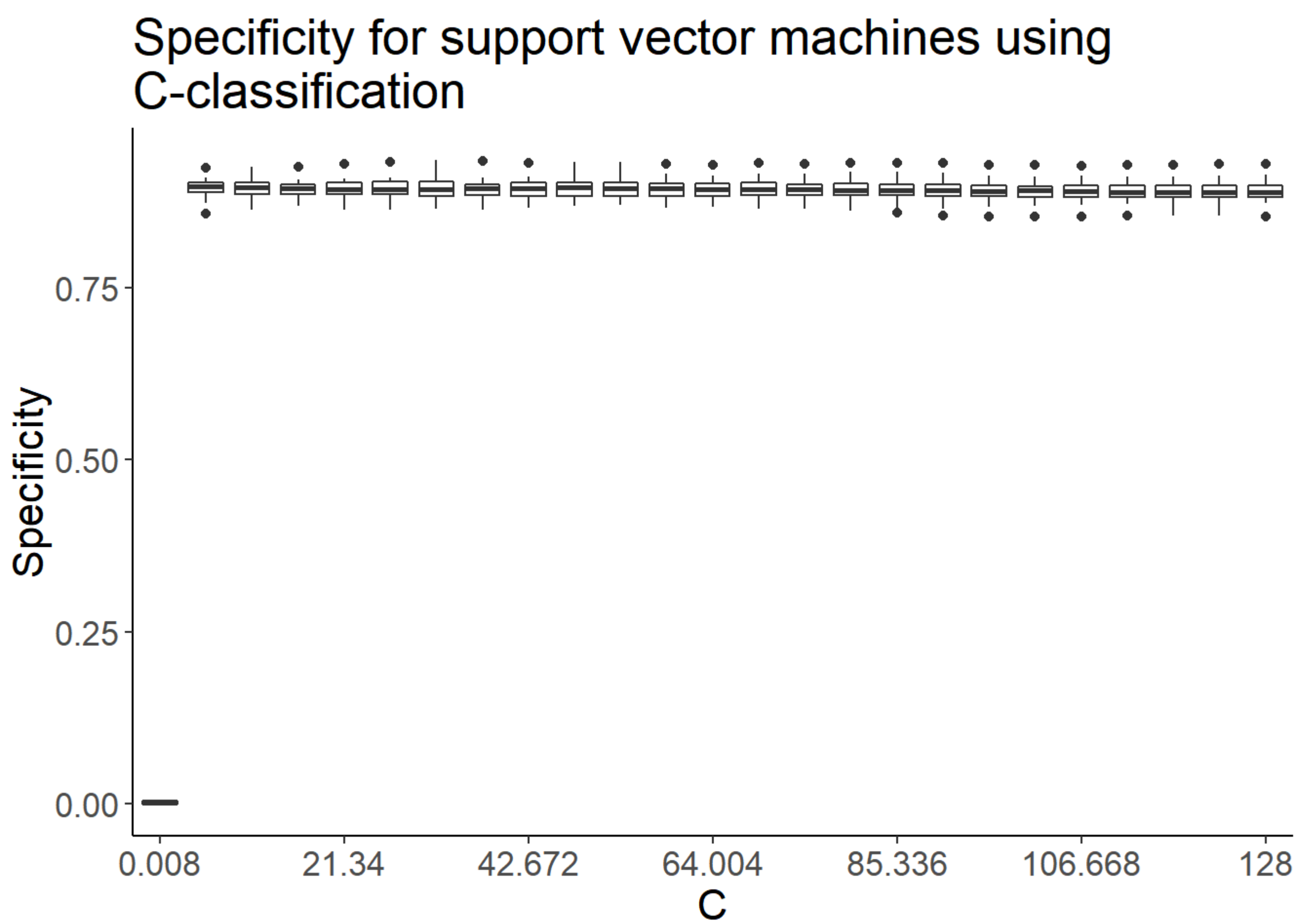
```r
# Plot accuracy for C-SVM
ggplot(data = err_mat_melt_opt_c_svm,
       aes(x = c_value, y = 1-Error)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c[1], v_c[5], v_c[9], v_c[13], v_c[17],
                              v_c[21], v_c[25]),
                   labels = c(round(v_c[1], 3), round(v_c[5], 3),
                              round(v_c[9], 3), round(v_c[13], 3),
                              round(v_c[17], 3), round(v_c[21], 3),
                              round(v_c[25], 3))) +
  labs(x = "C", title = "Accuracy for support vector machines using\nC-classification")
```



```r
# Plot sensitivity for C-SVM
ggplot(data = sens_mat_melt_opt_c_svm,
       aes(x = c_value, y = Sensitivity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c[1], v_c[5], v_c[9], v_c[13], v_c[17],
                              v_c[21], v_c[25]),
                   labels = c(round(v_c[1], 3), round(v_c[5], 3),
                              round(v_c[9], 3), round(v_c[13], 3),
                              round(v_c[17], 3), round(v_c[21], 3),
                              round(v_c[25], 3))) +
  labs(x = "C", title = "Sensitivity for support vector machines using\nC-classification")
```



```
# Plot specificity for C-SVM
ggplot(data = spec_mat_melt_opt_c_svm,
       aes(x = c_value, y = Specificity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c[1], v_c[5], v_c[9], v_c[13], v_c[17],
                              v_c[21], v_c[25]),
                   labels = c(round(v_c[1], 3), round(v_c[5], 3),
                              round(v_c[9], 3), round(v_c[13], 3),
                              round(v_c[17], 3), round(v_c[21], 3),
                              round(v_c[25], 3))) +
  labs(x = "C", title = "Specificity for support vector machines using\nC-classification")
```



Although none of the tested values of C produces models whose results match those of the optimized random forests, there are huge jumps in each metric between the first and second values of C. We will test intermediate values to see if a better C-SVM model exists.

Try more small values of C (rerun vs. random forests)

```
R = 50 # set the number of replications

# set up train control to do CV
fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

set.seed(1)

# Create sequence of values of mtry
num_var <- seq(1, 15, 1)

# Create sequence of values of ntree
num_trees <- c(100)

# Create sequence of weights
weights <- list(c(1, 1))

# create the error matrix to store values
err_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# create specificity matrix to store values
spec_mat_opt_rf = matrix(0,
                      ncol=length(num_var)*length(num_trees)*length(weights),
                         nrow=R)

# Set number of values of C to test
n_c <- 31

# Create sequence of values of C to test
v_c = seq(0.1, 6.1, length=n_c)

# create the error matrix to store values
err_mat_opt_c_svm = matrix(0,
                        ncol=n_c,
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_opt_c_svm = matrix(0,
                         ncol=n_c,
                         nrow=R)

# create specificity matrix to store values
```

```
spec_mat_opt_c_svm = matrix(0,
                         ncol=n_c,
                         nrow=R)

# Loop through the repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through values of mtry
  for (i in 1:length(num_var)) {
    # Loop through numbers of trees (not actually used here)
    for (j in 1:length(num_trees)) {
      # Loop through weights (not actually used here)
      for (k in 1:length(weights)) {
        # Run random forest model
        mod_rf = train(spam ~ .,
                       spam_train,
                       trControl = fitControl,
                       method = "rf",
                       tuneGrid = expand.grid(mtry = num_var[i]),
                       ntree = num_trees[j],
                       classwt = weights[[k]],
                       metric = "ROC")

        # Make predictions on test set
        yhat_rf = predict(mod_rf, spam_test[,-58])

        # Calculate the correct column for the output
        index <- (i-1)*length(num_trees)*length(weights) +
          (j-1)*length(weights) + k

        # Calculate error rate
        err_mat_opt_rf[r, index] =
          mean(yhat_rf!=spam_test[,58])

        # Calculate confusion matrix
        cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")

        # Calculate sensitivity
        sens_mat_opt_rf[r, index] =
          cm_rf$byClass["Sensitivity"]

        # Calculate specificity
        spec_mat_opt_rf[r, index] =
          cm_rf$byClass["Specificity"]
```

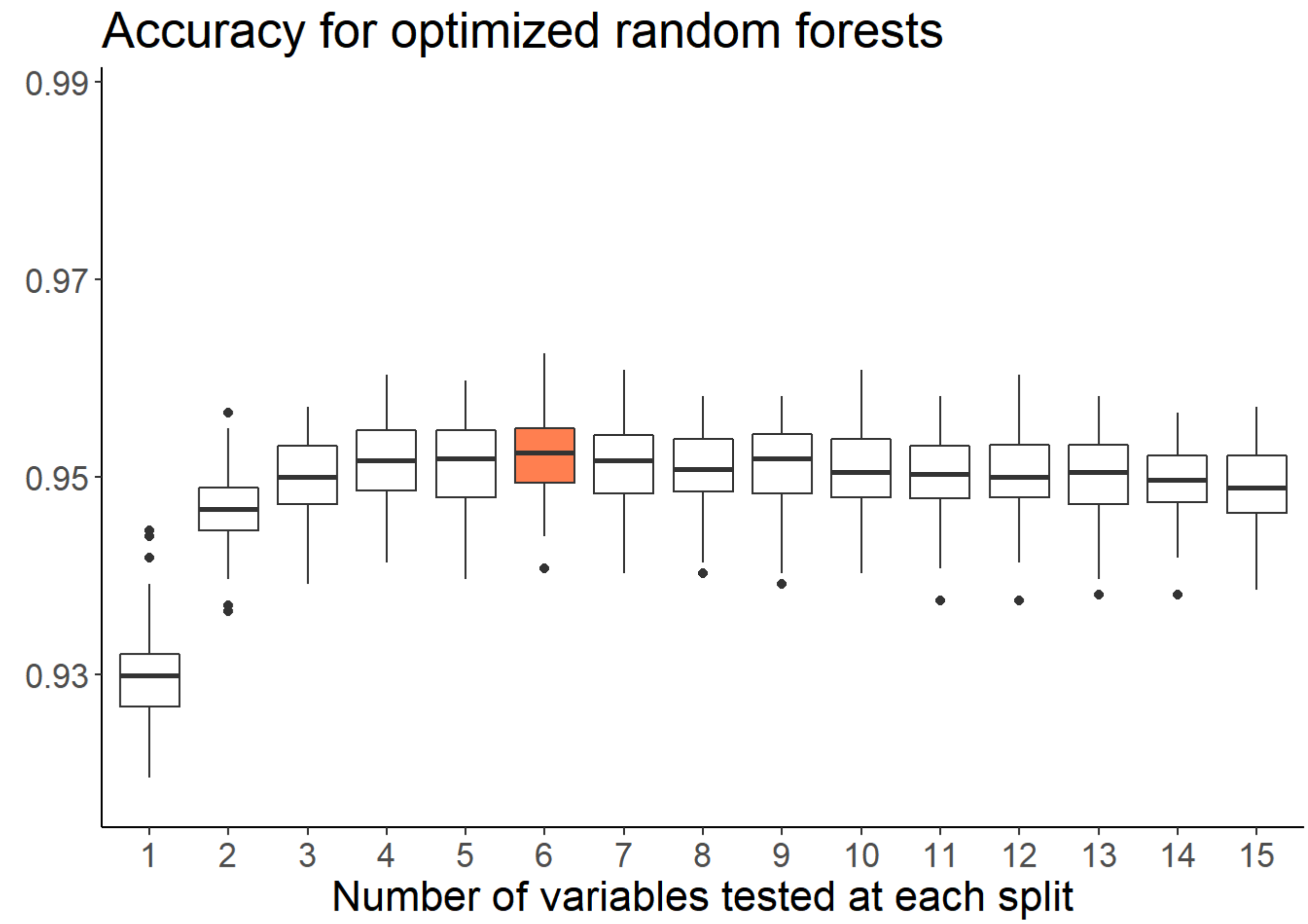
```
      }
    }
  }

  # Loop through values of C
  for(n in 1:n_c) {
    # Run C-SVM model
    mod_c_svm <- svm(spam~.,
                     spam_train,
                     cross=5,
                     cost=v_c[n],
                     type='C-classification',
                     metric = "ROC")
    # Make predictions on test set
    yhat_c_svm = predict(mod_c_svm, spam_test[,-58])
    # Calculate error rate
    err_mat_opt_c_svm[r,n] = mean(yhat_c_svm!=spam_test[,58])
    # Calculate confusion matrix
    cm_c_svm <- confusionMatrix(yhat_c_svm, spam_test[,58], positive = "No")
    # Calculate sensitivity
    sens_mat_opt_c_svm[r,n] = cm_c_svm$byClass["Sensitivity"]
    # Calculate specificity
    spec_mat_opt_c_svm[r,n] = cm_c_svm$byClass["Specificity"]
  }


  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

#Melt output to prepare for plotting
err_mat_melt_opt_rf = melt(as.data.frame(err_mat_opt_rf))
colnames(err_mat_melt_opt_rf) = c('Method','Error')

sens_mat_melt_opt_rf = melt(as.data.frame(sens_mat_opt_rf))
colnames(sens_mat_melt_opt_rf) = c('Method','Sensitivity')

spec_mat_melt_opt_rf = melt(as.data.frame(spec_mat_opt_rf))
colnames(spec_mat_melt_opt_rf) = c('Method','Specificity')

err_mat_melt_opt_c_svm = melt(as.data.frame(err_mat_opt_c_svm))
colnames(err_mat_melt_opt_c_svm) = c('Method','Error')

sens_mat_melt_opt_c_svm = melt(as.data.frame(sens_mat_opt_c_svm))
colnames(sens_mat_melt_opt_c_svm) = c('Method','Sensitivity')

spec_mat_melt_opt_c_svm = melt(as.data.frame(spec_mat_opt_c_svm))
colnames(spec_mat_melt_opt_c_svm) = c('Method','Specificity')
```

```
# Rename output to avoid conflicts
err_mat_melt_opt_rf2 <- err_mat_melt_opt_rf
sens_mat_melt_opt_rf2 <- sens_mat_melt_opt_rf
spec_mat_melt_opt_rf2 <- spec_mat_melt_opt_rf
err_mat_melt_opt_c_svm2 <- err_mat_melt_opt_c_svm
sens_mat_melt_opt_c_svm2 <- sens_mat_melt_opt_c_svm
spec_mat_melt_opt_c_svm2 <- spec_mat_melt_opt_c_svm
```

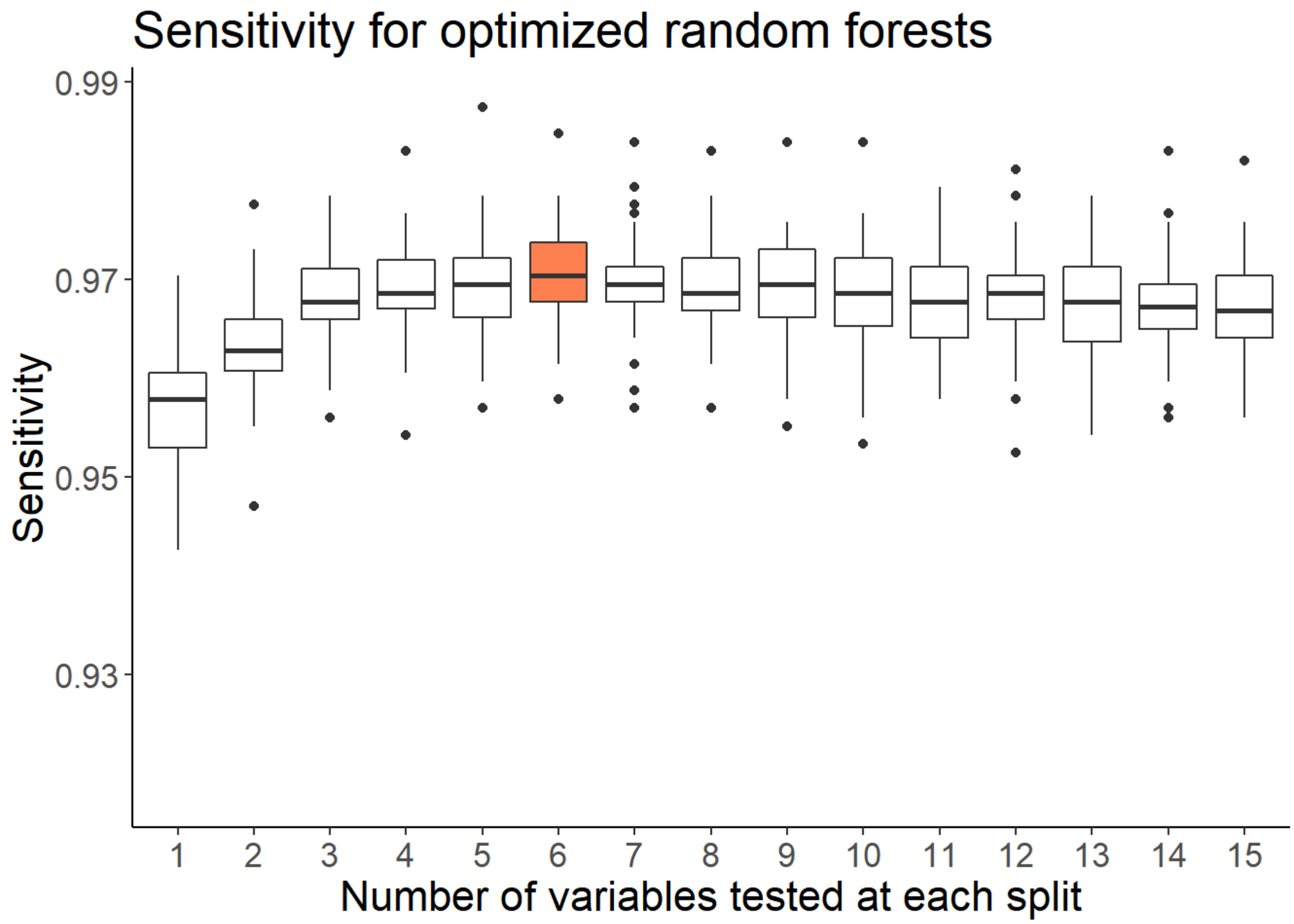
Graph Results

```
# Add column of mtry values to aid plotting
err_mat_melt_opt_rf2 <- err_mat_melt_opt_rf2 %>%
  mutate(mtry = as.factor(sort(rep(seq(1, 15), 50))))
sens_mat_melt_opt_rf2 <- sens_mat_melt_opt_rf2 %>%
  mutate(mtry = as.factor(sort(rep(seq(1, 15), 50))))
spec_mat_melt_opt_rf2 <- spec_mat_melt_opt_rf2 %>%
  mutate(mtry = as.factor(sort(rep(seq(1, 15), 50))))
```

```
# Plot accuracy for optimized random forests
ggplot(data = err_mat_melt_opt_rf2,
       aes(x = mtry, y = 1-Error,
           fill = factor(ifelse(mtry==6,"Highlighted","Normal")))
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16),
        axis.title.y = element_blank()) +
  labs(x = "Number of variables tested at each split", title = "Accuracy for optimized random fo
rests", y = "Accuracy") +
  scale_y_continuous(limits = c(0.918, 0.988)) +
  scale_fill_manual(values=c("#FF7F50","white"))
```

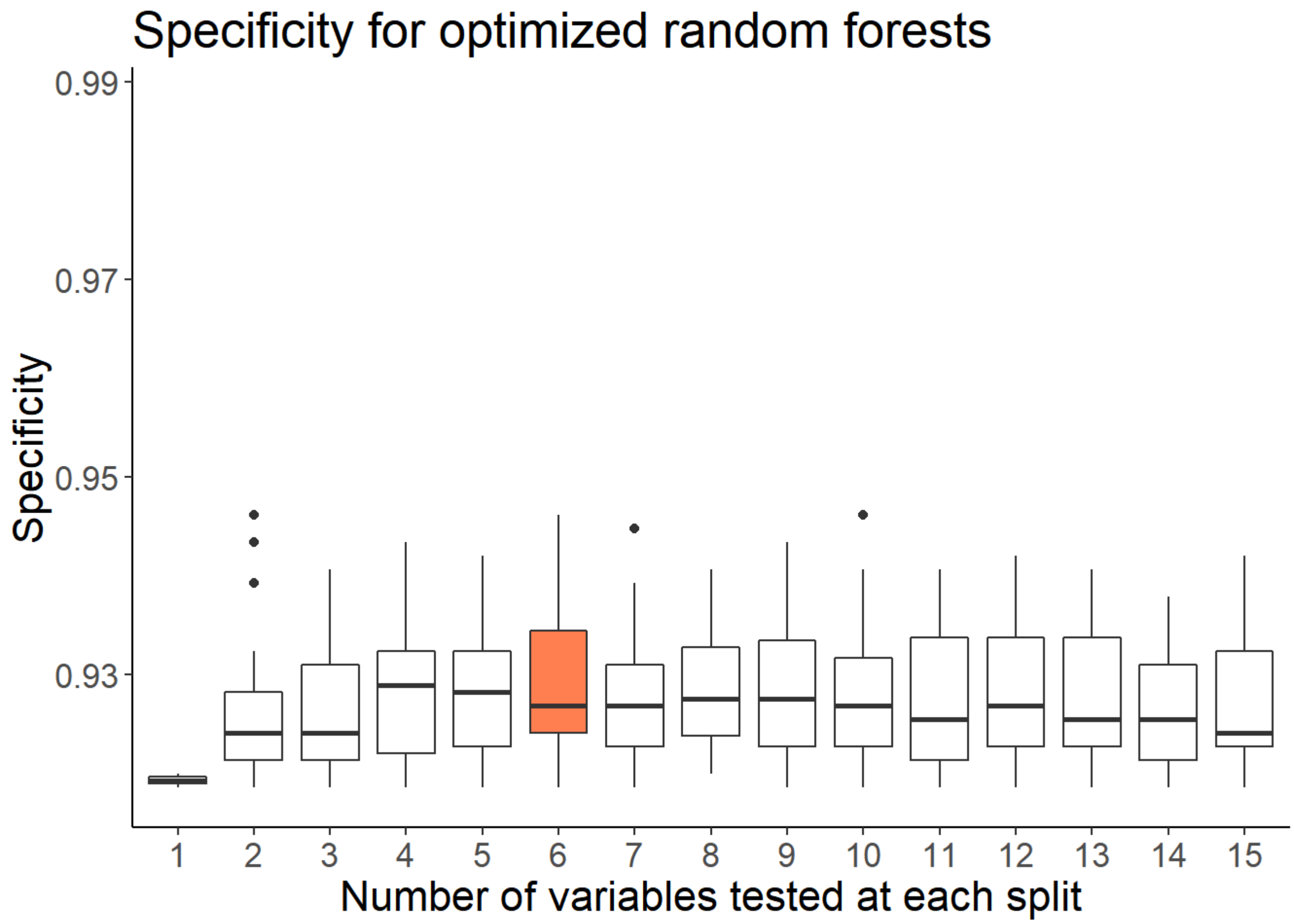



```
# Plot sensitivity for optimized random forests
ggplot(data = sens_mat_melt_opt_rf2,
       aes(x = mtry, y = Sensitivity,
           fill = factor(ifelse(mtry==6,"Highlighted","Normal")))
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Number of variables tested at each split", title = "Sensitivity for optimized random
 forests") +
  scale_y_continuous(limits = c(0.918, 0.988)) +
  scale_fill_manual(values=c("#FF7F50","white"))
```




```
# Plot specificity for optimized random forests
ggplot(data = spec_mat_melt_opt_rf2,
       aes(x = mtry, y = Specificity,
           fill = factor(ifelse(mtry==6,"Highlighted","Normal")))
       ) +
  geom_boxplot(show.legend = FALSE) +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  labs(x = "Number of variables tested at each split", title = "Specificity for optimized random
 forests") +
  scale_y_continuous(limits = c(0.918, 0.988)) +
  scale_fill_manual(values=c("#FF7F50","white"))
```

```
## Warning: Removed 266 rows containing non-finite values (stat_boxplot).
```

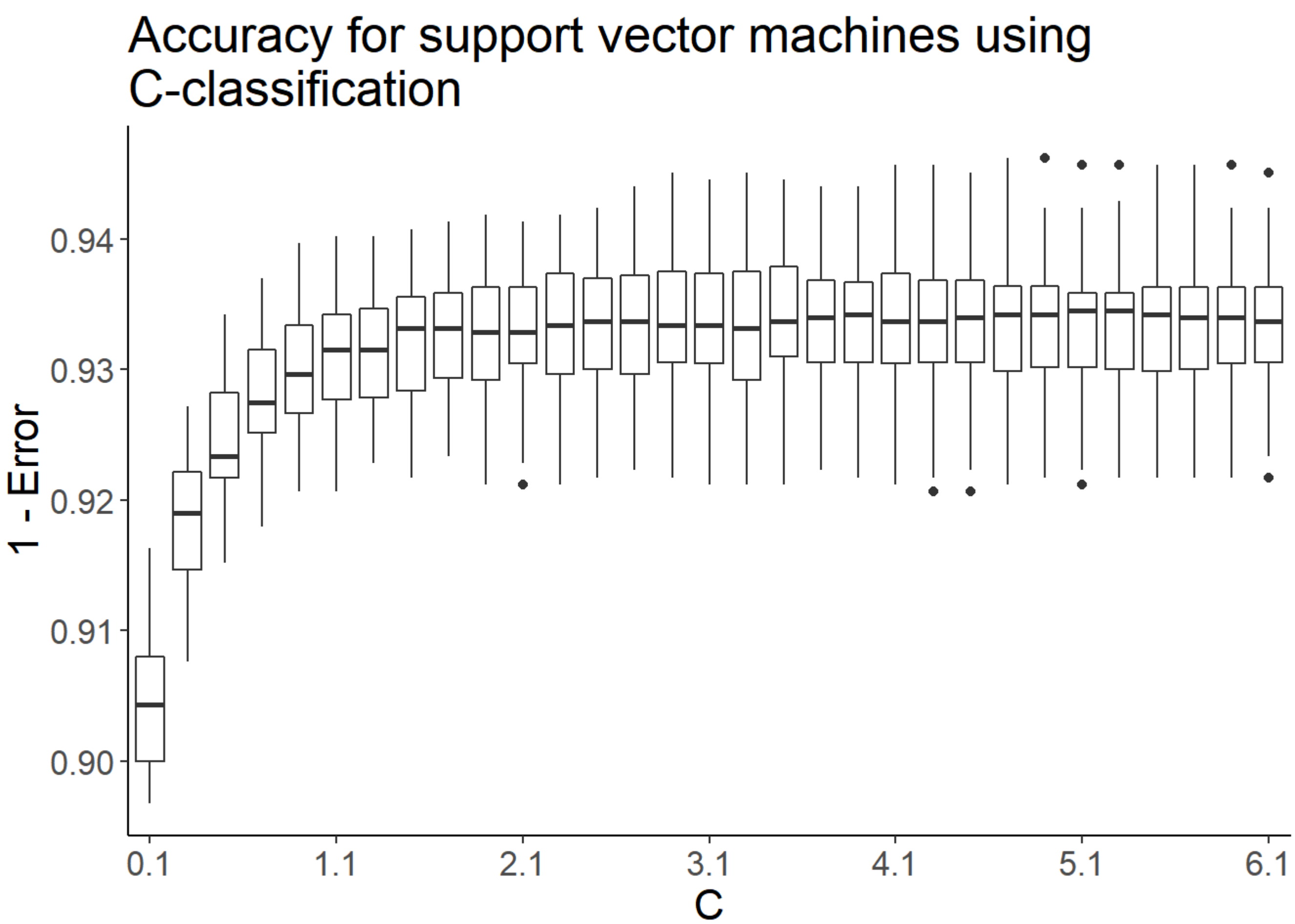


Mtry 6 did the best. Check the C results too.

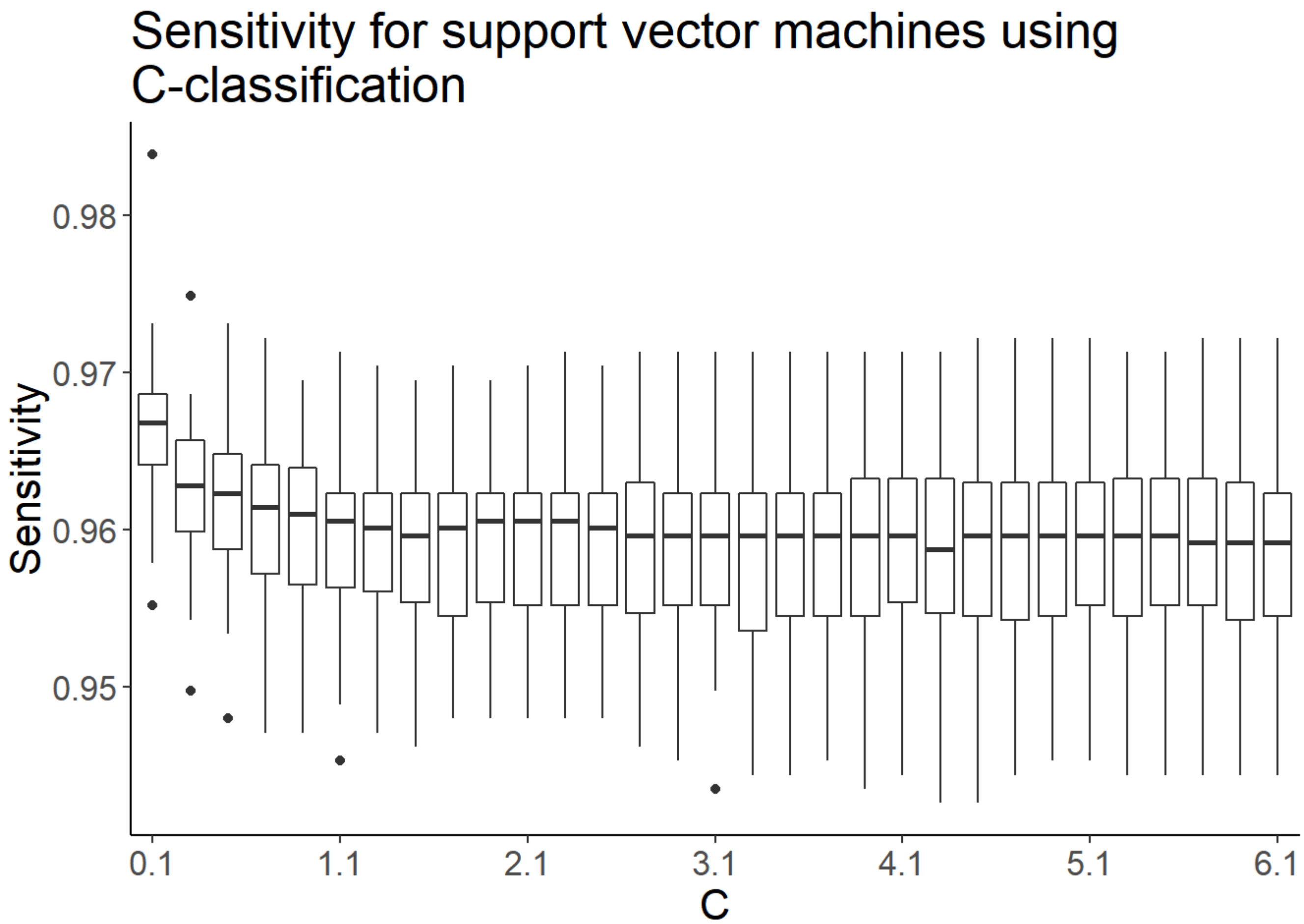
```
# Set number of values of C tested
n_c_2 <- 31

# Create sequence of values of C tested
v_c_2 = seq(0.1, 6.1, length=n_c_2)
```

```
# Add column of values of C to aid plotting
err_mat_melt_opt_c_svm2 <- err_mat_melt_opt_c_svm2 %>%
  mutate(c_value = as.factor(sort(rep(v_c_2, 50))))
sens_mat_melt_opt_c_svm2 <- sens_mat_melt_opt_c_svm2 %>%
  mutate(c_value = as.factor(sort(rep(v_c_2, 50))))
spec_mat_melt_opt_c_svm2 <- spec_mat_melt_opt_c_svm2 %>%
  mutate(c_value = as.factor(sort(rep(v_c_2, 50))))
```

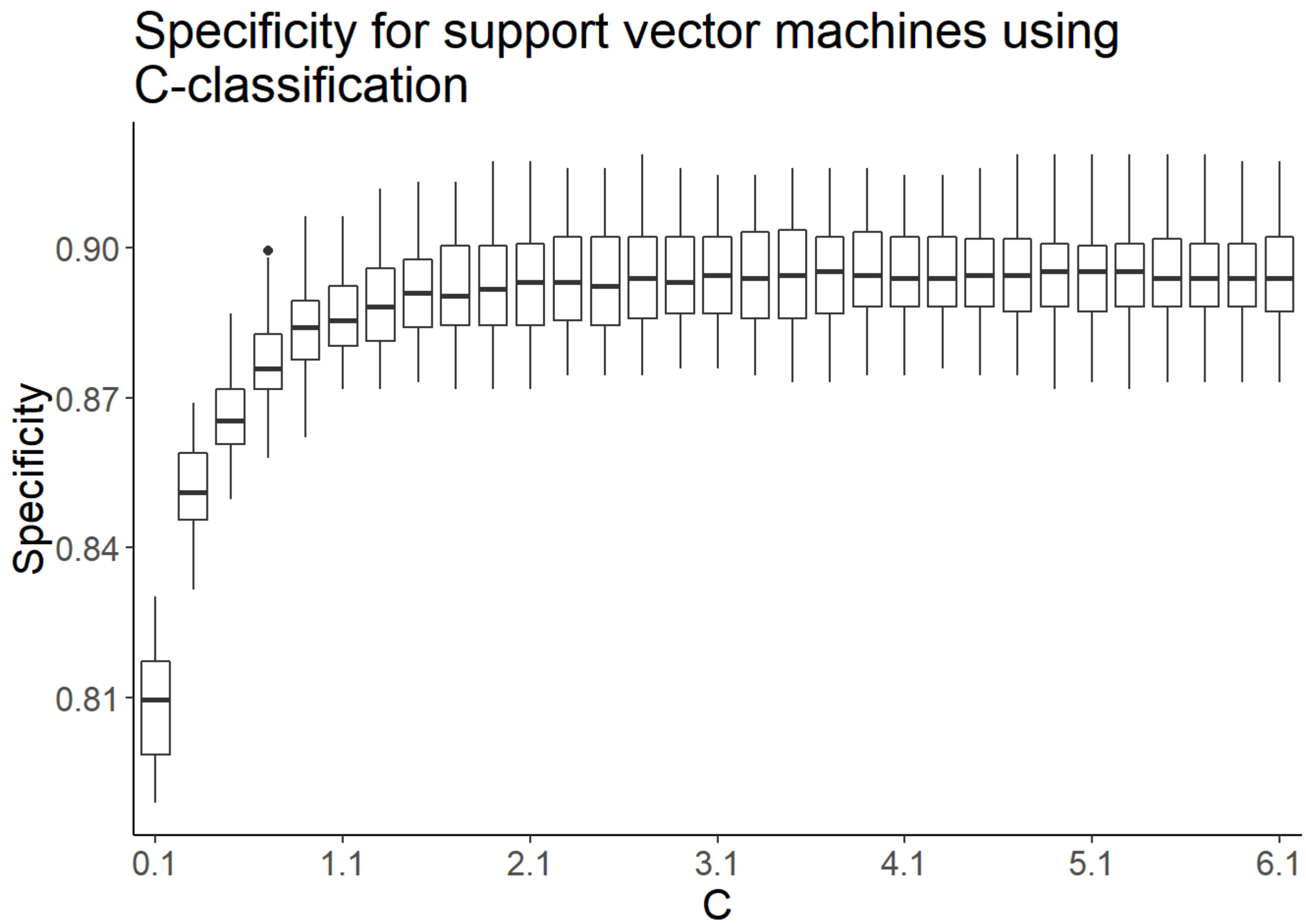
```
# Plot accuracy for C-SVM
ggplot(data = err_mat_melt_opt_c_svm2,
       aes(x = c_value, y = 1-Error)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c_2[1], v_c_2[6], v_c_2[11], v_c_2[16],
                              v_c_2[21], v_c_2[26], v_c_2[31])) +
  labs(x = "C", title = "Accuracy for support vector machines using\nC-classification")
```




```
# Plot sensitivity for C-SVM
ggplot(data = sens_mat_melt_opt_c_svm2,
       aes(x = c_value, y = Sensitivity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c_2[1], v_c_2[6], v_c_2[11], v_c_2[16],
                              v_c_2[21], v_c_2[26], v_c_2[31])) +
  labs(x = "C", title = "Sensitivity for support vector machines using\nC-classification")
```




```
# Plot specificity for C-SVM
ggplot(data = spec_mat_melt_opt_c_svm2,
       aes(x = c_value, y = Specificity)
       ) +
  geom_boxplot() +
  theme_classic() +
  theme(text = element_text(size = 16)) +
  scale_x_discrete(breaks = c(v_c_2[1], v_c_2[6], v_c_2[11], v_c_2[16],
                              v_c_2[21], v_c_2[26], v_c_2[31])) +
  labs(x = "C", title = "Specificity for support vector machines using\nC-classification")
```



None of the C-SVM models matches the optimized random forests. However, we should also kernalized SVM.

Why not! KSVM

```r
R = 20 # set the number of replications

set.seed(1)

# Set number of values of C to test
n_c <- 25

# Create sequence of values of C to test
v_c = seq(0.1, 15, length=n_c)

# create the error matrix to store values
err_mat_rbf = matrix(0,
                     ncol=n_c,
                     nrow=R)

# create sensitivity matrix to store values
sens_mat_rbf = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create specificity matrix to store values
spec_mat_rbf = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create the error matrix to store values
err_mat_poly = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create sensitivity matrix to store values
sens_mat_poly = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create specificity matrix to store values
spec_mat_poly = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create the error matrix to store values
err_mat_van = matrix(0,
                     ncol=n_c,
                     nrow=R)

# create sensitivity matrix to store values
sens_mat_van = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create specificity matrix to store values
spec_mat_van = matrix(0,
                      ncol=n_c,
```

```
                      nrow=R)

# create the error matrix to store values
err_mat_tanh = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create sensitivity matrix to store values
sens_mat_tanh = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create specificity matrix to store values
spec_mat_tanh = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create the error matrix to store values
err_mat_lap = matrix(0,
                     ncol=n_c,
                     nrow=R)

# create sensitivity matrix to store values
sens_mat_lap = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create specificity matrix to store values
spec_mat_lap = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create the error matrix to store values
err_mat_bess = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create sensitivity matrix to store values
sens_mat_bess = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create specificity matrix to store values
spec_mat_bess = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create the error matrix to store values
err_mat_anova = matrix(0,
                       ncol=n_c,
                       nrow=R)

# create sensitivity matrix to store values
sens_mat_anova = matrix(0,
```

```
                        ncol=n_c,
                        nrow=R)

# create specificity matrix to store values
spec_mat_anova = matrix(0,
                        ncol=n_c,
                        nrow=R)

# create the error matrix to store values
err_mat_spline = matrix(0,
                        ncol=n_c,
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_spline = matrix(0,
                         ncol=n_c,
                         nrow=R)

# create specificity matrix to store values
spec_mat_spline = matrix(0,
                         ncol=n_c,
                         nrow=R)

# create the error matrix to store values
err_mat_string = matrix(0,
                        ncol=n_c,
                        nrow=R)

# create sensitivity matrix to store values
sens_mat_string = matrix(0,
                         ncol=n_c,
                         nrow=R)

# create specificity matrix to store values
spec_mat_string = matrix(0,
                         ncol=n_c,
                         nrow=R)

for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through values of C
  for(n in 1:n_c) {
    # Run k-SVM with radial basis kernel, predict, and calculate metrics
    mod_rbf <- ksvm(spam~.,
                    spam_train,
```

```
                cross=0,
                C=v_c[n],
                kernel = "rbfdot",
                type='C-svc',
                metric = "ROC")
yhat_rbf = predict(mod_rbf, spam_test[,-58])
err_mat_rbf[r,n] = mean(yhat_rbf!=spam_test[,58])
cm_rbf <- confusionMatrix(yhat_rbf, spam_test[,58], positive = "No")
sens_mat_rbf[r,n] = cm_rbf$byClass["Sensitivity"]
spec_mat_rbf[r,n] = cm_rbf$byClass["Specificity"]

# Run k-SVM with polynomial kernel, predict, and calculate metrics
mod_poly <- ksvm(spam~.,
                spam_train,
                cross=0,
                C=v_c[n],
                kernel = "polydot",
                type='C-svc',
                metric = "ROC")
yhat_poly = predict(mod_poly, spam_test[,-58])
err_mat_poly[r,n] = mean(yhat_poly!=spam_test[,58])
cm_poly <- confusionMatrix(yhat_poly, spam_test[,58], positive = "No")
sens_mat_poly[r,n] = cm_poly$byClass["Sensitivity"]
spec_mat_poly[r,n] = cm_poly$byClass["Specificity"]

# Run k-SVM with linear kernel, predict, and calculate metrics
mod_van <- ksvm(spam~.,
                spam_train,
                cross=0,
                C=v_c[n],
                kernel = "vanilladot",
                type='C-svc',
                metric = "ROC")
yhat_van = predict(mod_van, spam_test[,-58])
err_mat_van[r,n] = mean(yhat_van!=spam_test[,58])
cm_van <- confusionMatrix(yhat_van, spam_test[,58], positive = "No")
sens_mat_van[r,n] = cm_van$byClass["Sensitivity"]
spec_mat_van[r,n] = cm_van$byClass["Specificity"]

# Run k-SVM with hyperbolic tangent kernel, predict, and calculate metrics
mod_tanh <- ksvm(spam~.,
                spam_train,
                cross=0,
                C=v_c[n],
                kernel = "tanhdot",
                type='C-svc',
                metric = "ROC")
yhat_tanh = predict(mod_tanh, spam_test[,-58])
err_mat_tanh[r,n] = mean(yhat_tanh!=spam_test[,58])
cm_tanh <- confusionMatrix(yhat_tanh, spam_test[,58], positive = "No")
sens_mat_tanh[r,n] = cm_tanh$byClass["Sensitivity"]
spec_mat_tanh[r,n] = cm_tanh$byClass["Specificity"]

# Run k-SVM with Laplacian kernel, predict, and calculate metrics
```

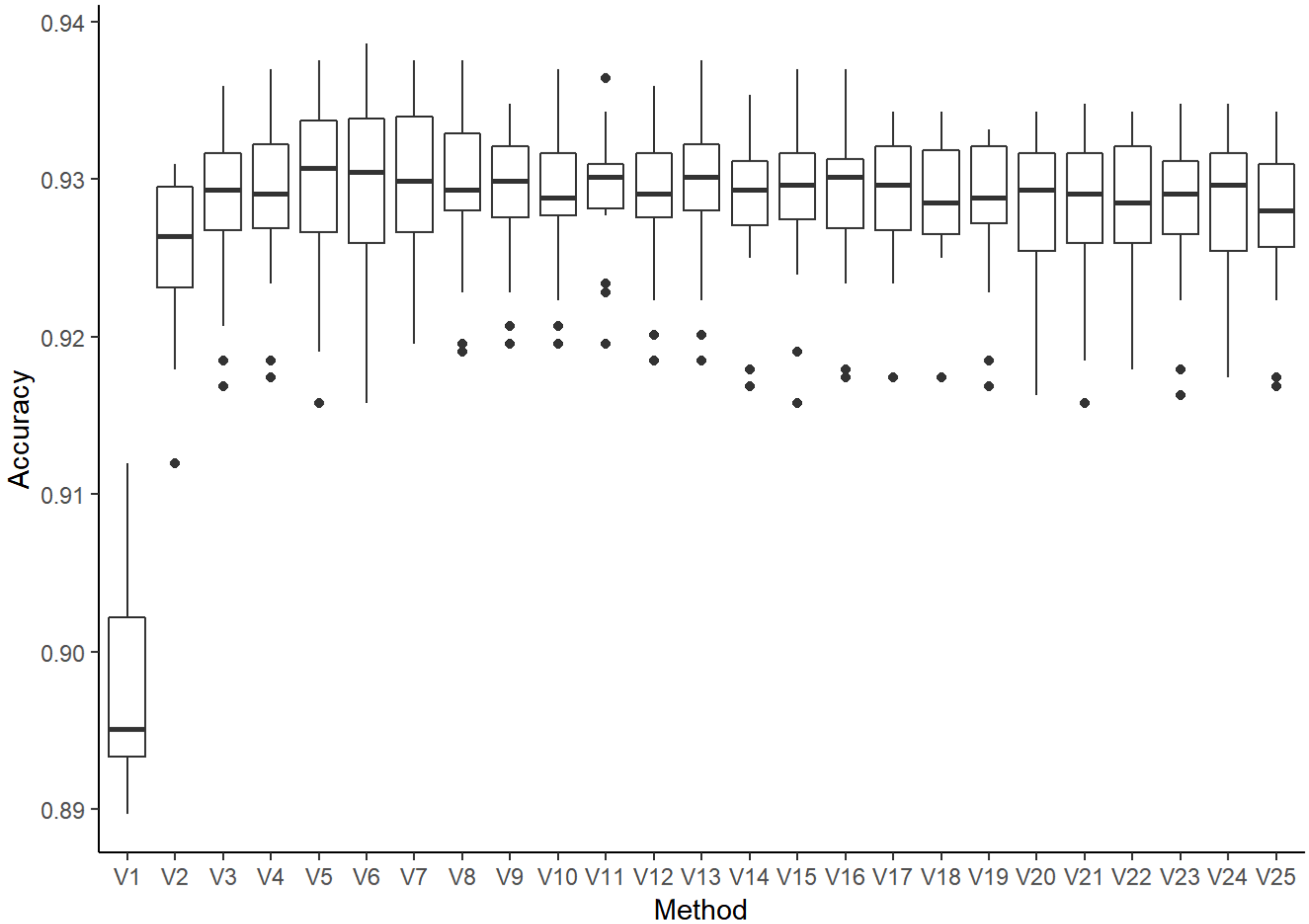
```
mod_lap <- ksvm(spam~.,
               spam_train,
               cross=0,
               C=v_c[n],
               kernel = "laplacedot",
               type='C-svc',
               metric = "ROC")
yhat_lap = predict(mod_lap, spam_test[,-58])
err_mat_lap[r,n] = mean(yhat_lap!=spam_test[,58])
cm_lap <- confusionMatrix(yhat_lap, spam_test[,58], positive = "No")
sens_mat_lap[r,n] = cm_lap$byClass["Sensitivity"]
spec_mat_lap[r,n] = cm_lap$byClass["Specificity"]

# Run k-SVM with Bessel kernel, predict, and calculate metrics
mod_bess <- ksvm(spam~.,
               spam_train,
               cross=0,
               C=v_c[n],
               kernel = "besseldot",
               type='C-svc',
               metric = "ROC")
yhat_bess = predict(mod_bess, spam_test[,-58])
err_mat_bess[r,n] = mean(yhat_bess!=spam_test[,58])
cm_bess <- confusionMatrix(yhat_bess, spam_test[,58], positive = "No")
sens_mat_bess[r,n] = cm_bess$byClass["Sensitivity"]
spec_mat_bess[r,n] = cm_bess$byClass["Specificity"]

# Run k-SVM with ANOVA RBF kernel, predict, and calculate metrics
mod_anova <- ksvm(spam~.,
               spam_train,
               cross=0,
               C=v_c[n],
               kernel = "anovadot",
               type='C-svc',
               metric = "ROC")
yhat_anova = predict(mod_anova, spam_test[,-58])
err_mat_anova[r,n] = mean(yhat_anova!=spam_test[,58])
cm_anova <- confusionMatrix(yhat_anova, spam_test[,58], positive = "No")
sens_mat_anova[r,n] = cm_anova$byClass["Sensitivity"]
spec_mat_anova[r,n] = cm_anova$byClass["Specificity"]

# Run k-SVM with spline kernel, predict, and calculate metrics
mod_spline <- ksvm(spam~.,
               spam_train,
               cross=0,
               C=v_c[n],
               kernel = "splinedot",
               type='C-svc',
               metric = "ROC")
yhat_spline = predict(mod_spline, spam_test[,-58])
err_mat_spline[r,n] = mean(yhat_spline!=spam_test[,58])
cm_spline <- confusionMatrix(yhat_spline, spam_test[,58], positive = "No")
sens_mat_spline[r,n] = cm_spline$byClass["Sensitivity"]
spec_mat_spline[r,n] = cm_spline$byClass["Specificity"]
```

```
  }


  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_rbf = melt(as.data.frame(err_mat_rbf))
colnames(err_mat_melt_rbf) = c('Method','Error')

sens_mat_melt_rbf = melt(as.data.frame(sens_mat_rbf))
colnames(sens_mat_melt_rbf) = c('Method','Sensitivity')

spec_mat_melt_rbf = melt(as.data.frame(spec_mat_rbf))
colnames(spec_mat_melt_rbf) = c('Method','Specificity')

err_mat_melt_poly = melt(as.data.frame(err_mat_poly))
colnames(err_mat_melt_poly) = c('Method','Error')

sens_mat_melt_poly = melt(as.data.frame(sens_mat_poly))
colnames(sens_mat_melt_poly) = c('Method','Sensitivity')

spec_mat_melt_poly = melt(as.data.frame(spec_mat_poly))
colnames(spec_mat_melt_poly) = c('Method','Specificity')

err_mat_melt_van = melt(as.data.frame(err_mat_van))
colnames(err_mat_melt_van) = c('Method','Error')

sens_mat_melt_van = melt(as.data.frame(sens_mat_van))
colnames(sens_mat_melt_van) = c('Method','Sensitivity')

spec_mat_melt_van = melt(as.data.frame(spec_mat_van))
colnames(spec_mat_melt_van) = c('Method','Specificity')

err_mat_melt_tanh = melt(as.data.frame(err_mat_tanh))
colnames(err_mat_melt_tanh) = c('Method','Error')

sens_mat_melt_tanh = melt(as.data.frame(sens_mat_tanh))
colnames(sens_mat_melt_tanh) = c('Method','Sensitivity')

spec_mat_melt_tanh = melt(as.data.frame(spec_mat_tanh))
colnames(spec_mat_melt_tanh) = c('Method','Specificity')

err_mat_melt_lap = melt(as.data.frame(err_mat_lap))
colnames(err_mat_melt_lap) = c('Method','Error')

sens_mat_melt_lap = melt(as.data.frame(sens_mat_lap))
colnames(sens_mat_melt_lap) = c('Method','Sensitivity')

spec_mat_melt_lap = melt(as.data.frame(spec_mat_lap))
colnames(spec_mat_melt_lap) = c('Method','Specificity')

err_mat_melt_bess = melt(as.data.frame(err_mat_bess))
```

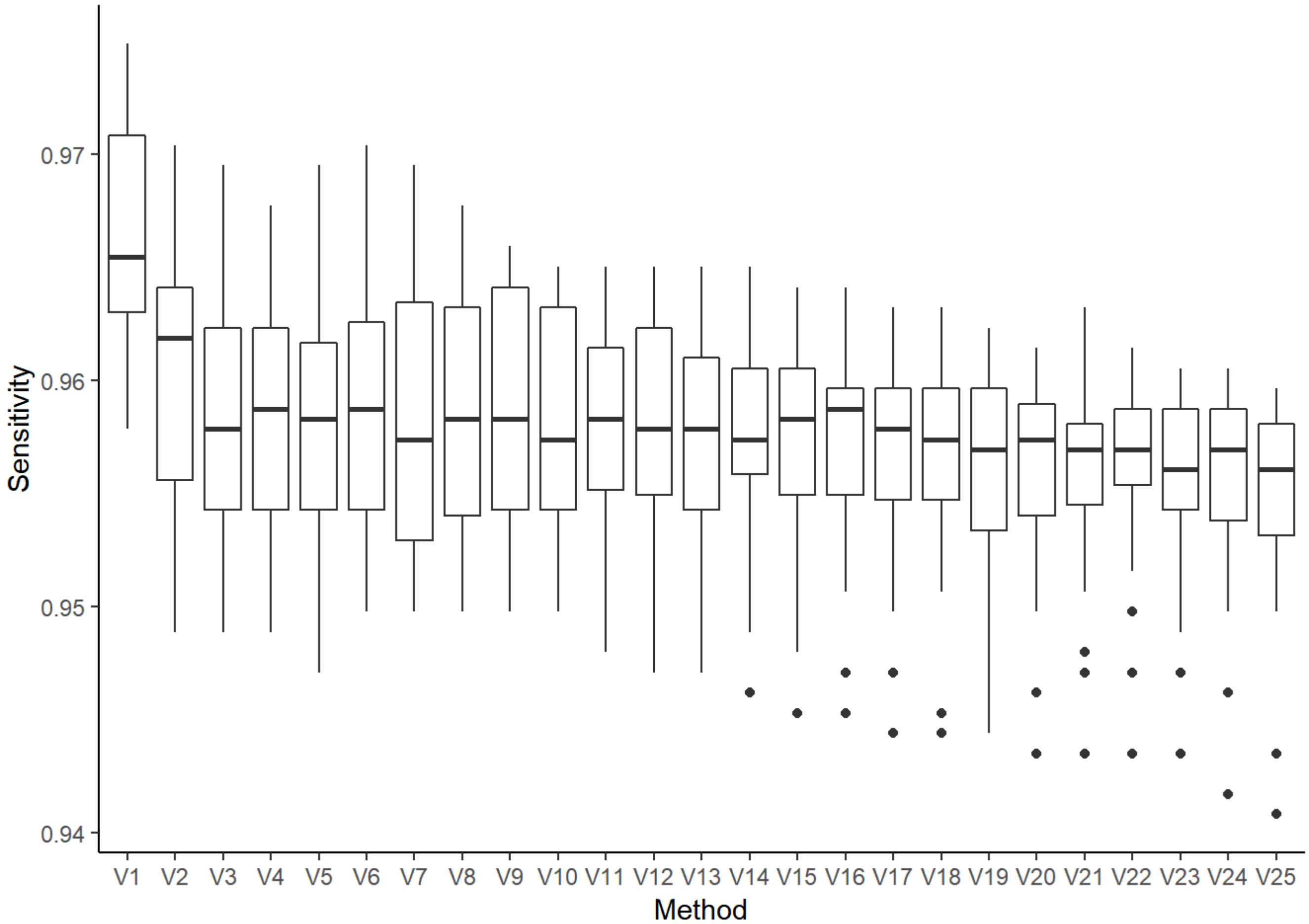
```
colnames(err_mat_melt_bess) = c('Method','Error')

sens_mat_melt_bess = melt(as.data.frame(sens_mat_bess))
colnames(sens_mat_melt_bess) = c('Method','Sensitivity')

spec_mat_melt_bess = melt(as.data.frame(spec_mat_bess))
colnames(spec_mat_melt_bess) = c('Method','Specificity')

err_mat_melt_anova = melt(as.data.frame(err_mat_anova))
colnames(err_mat_melt_anova) = c('Method','Error')

sens_mat_melt_anova = melt(as.data.frame(sens_mat_anova))
colnames(sens_mat_melt_anova) = c('Method','Sensitivity')

spec_mat_melt_anova = melt(as.data.frame(spec_mat_anova))
colnames(spec_mat_melt_anova) = c('Method','Specificity')

err_mat_melt_spline = melt(as.data.frame(err_mat_spline))
colnames(err_mat_melt_spline) = c('Method','Error')

sens_mat_melt_spline = melt(as.data.frame(sens_mat_spline))
colnames(sens_mat_melt_spline) = c('Method','Sensitivity')

spec_mat_melt_spline = melt(as.data.frame(spec_mat_spline))
colnames(spec_mat_melt_spline) = c('Method','Specificity')
```

Plot KSVM results

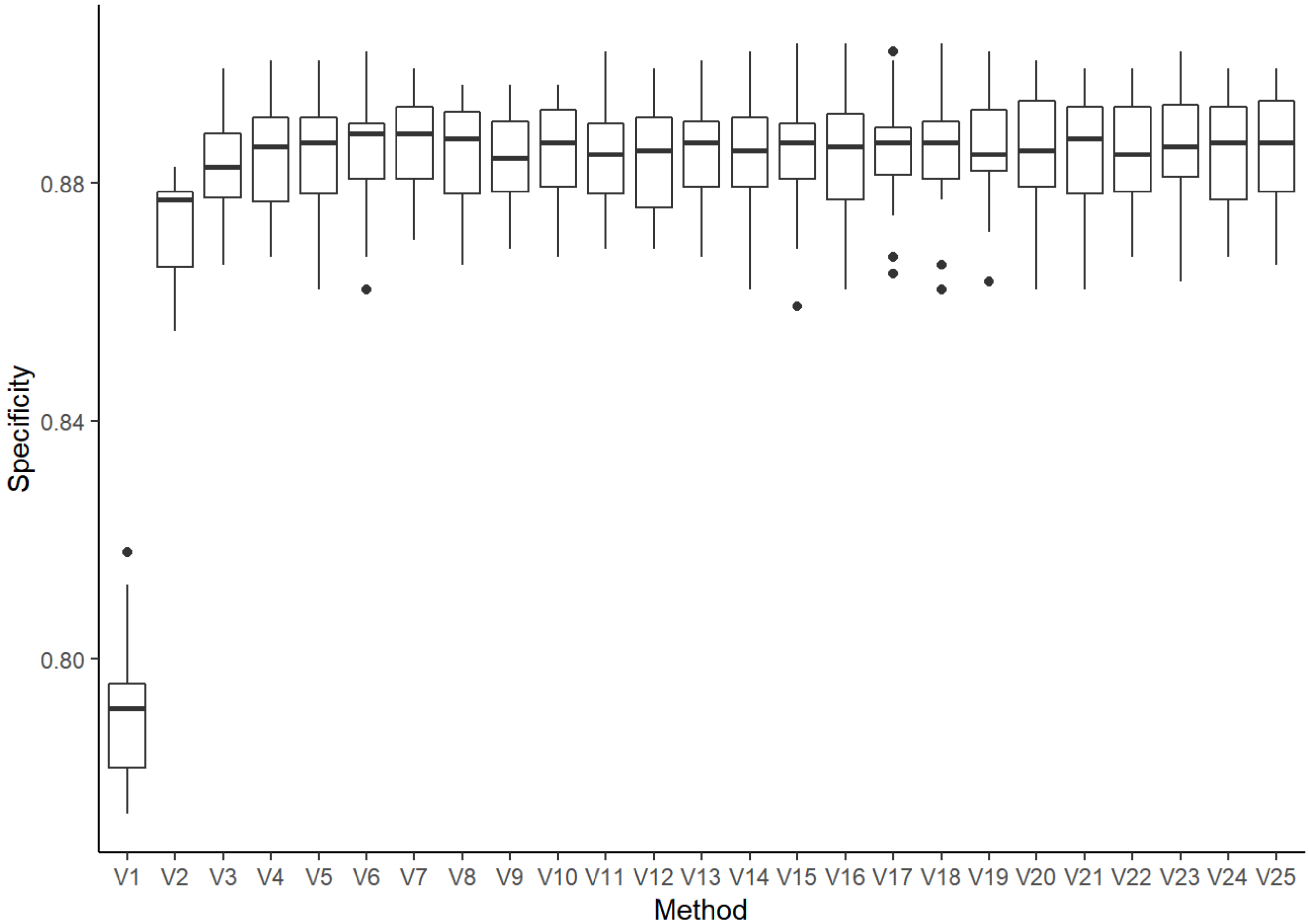
```
# Plot accuracy for k-SVM with radial basis kernel
ggplot(err_mat_melt_rbf,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic() + ylab("Accuracy")
```



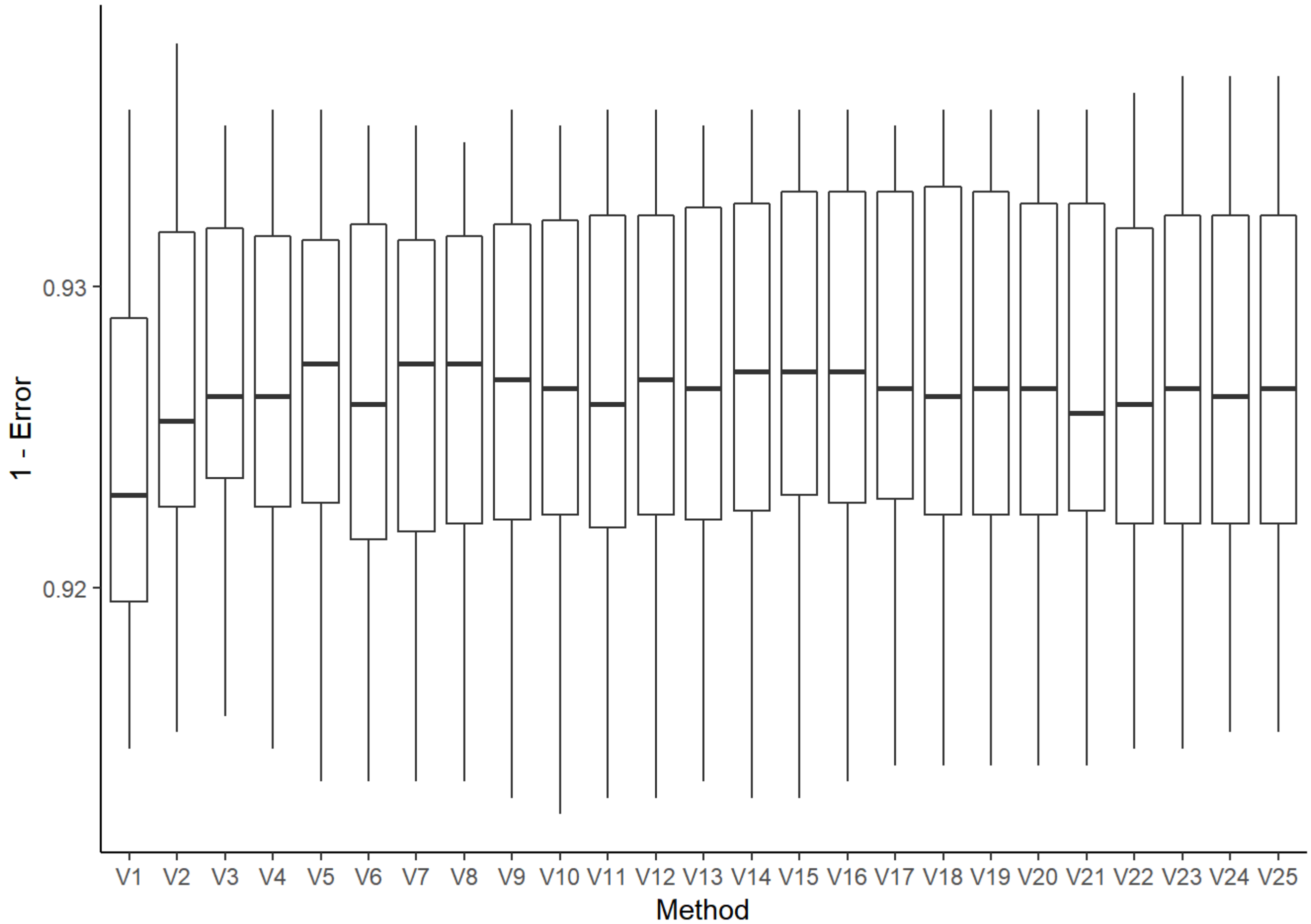

```
# Plot sensitivity for k-SVM with radial basis kernel
ggplot(sens_mat_melt_rbf,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



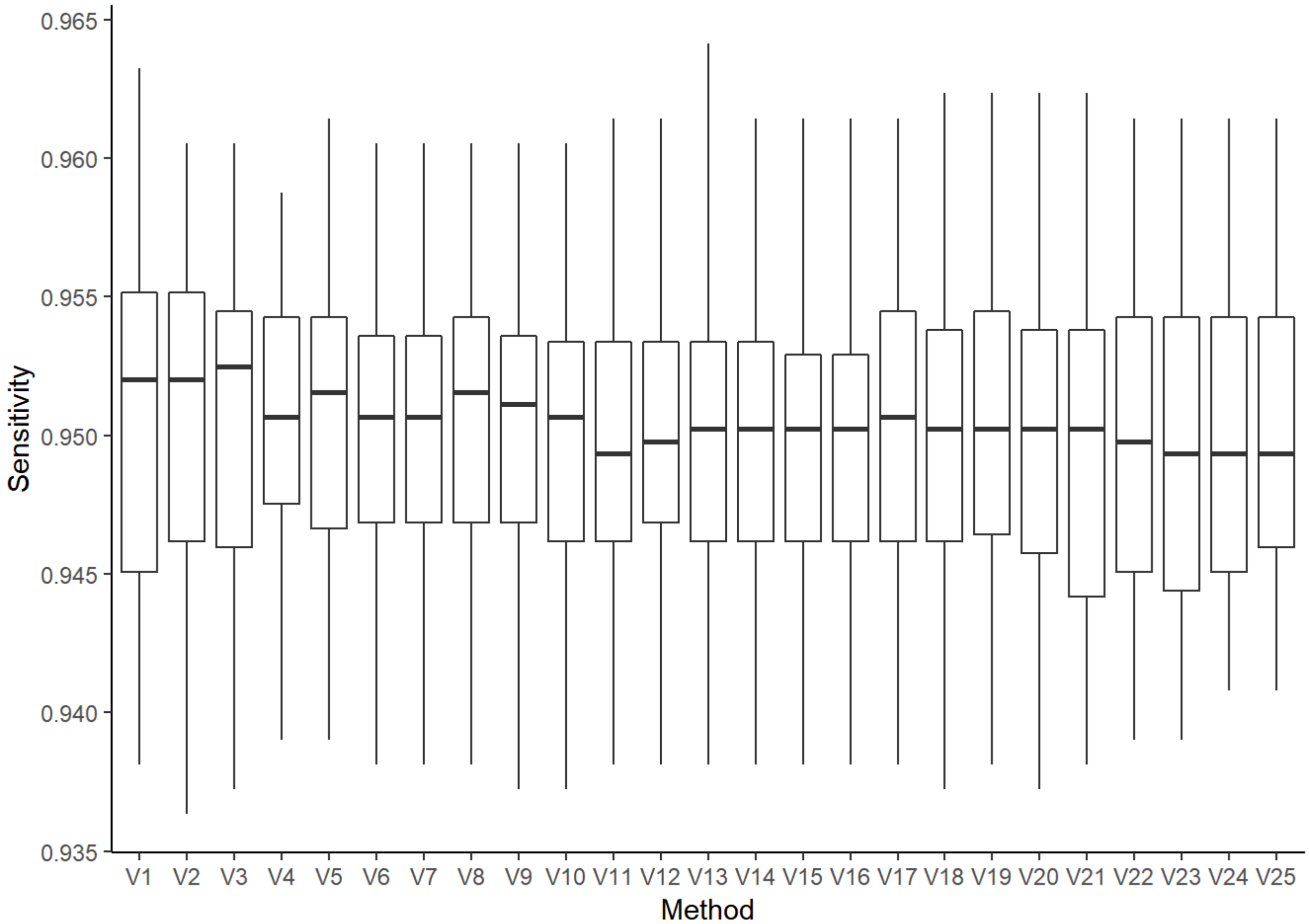

```r
# Plot specificity for k-SVM with radial basis kernel
ggplot(spec_mat_melt_rbf,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



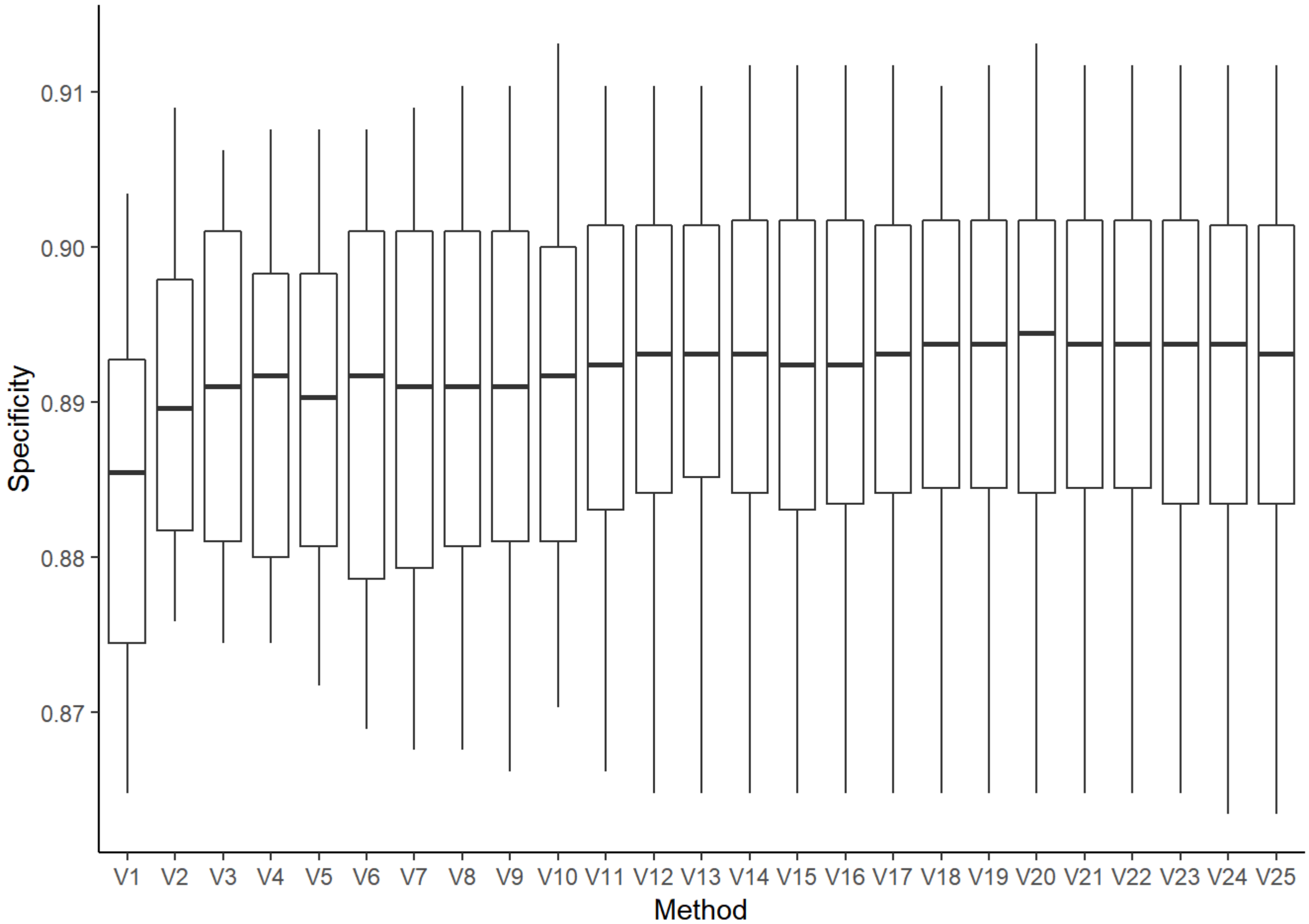

```
# Plot accuracy for k-SVM with polynomial kernel
ggplot(err_mat_melt_poly,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



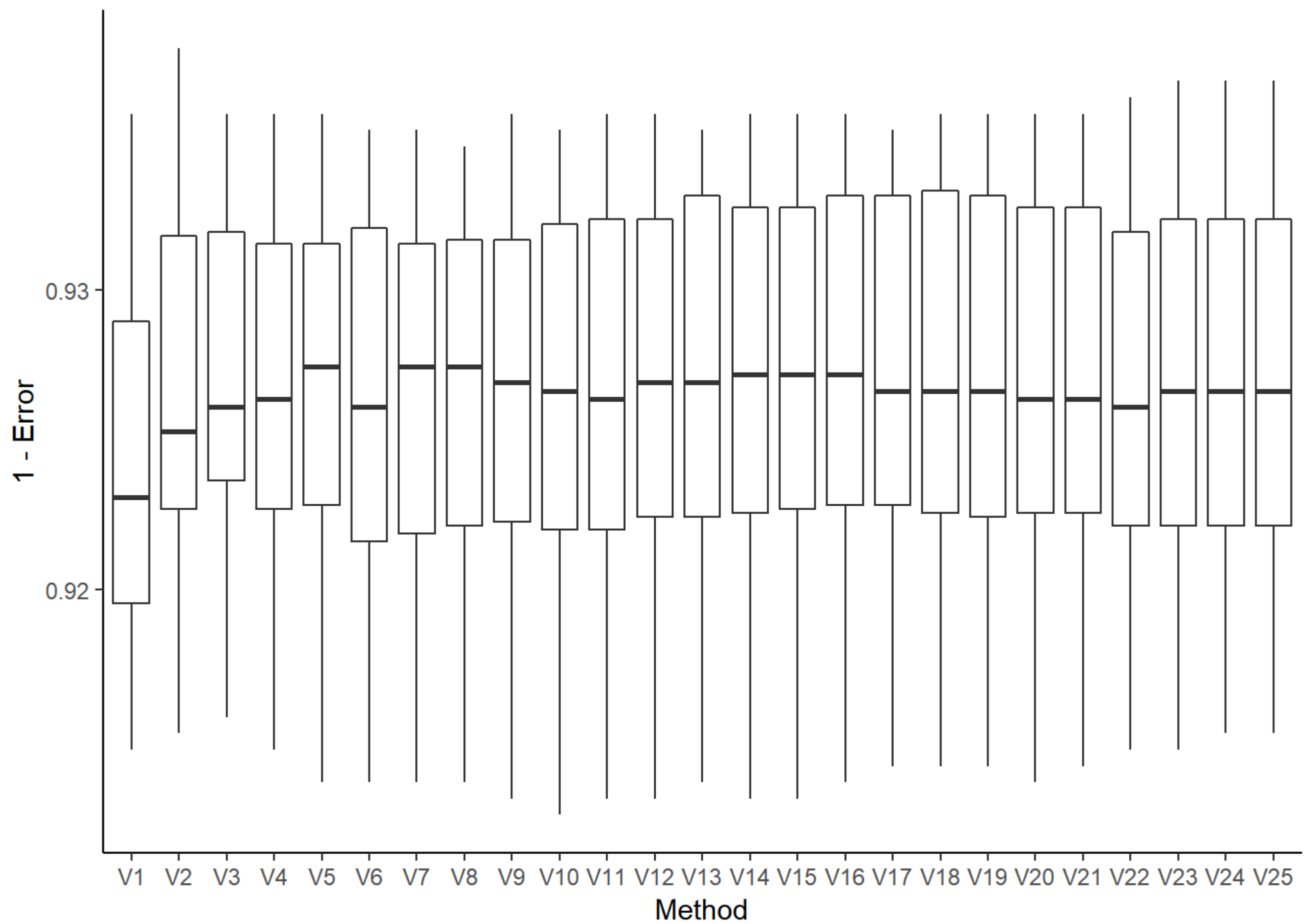

```
# Plot sensitivity for k-SVM with polynomial kernel
ggplot(sens_mat_melt_poly,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



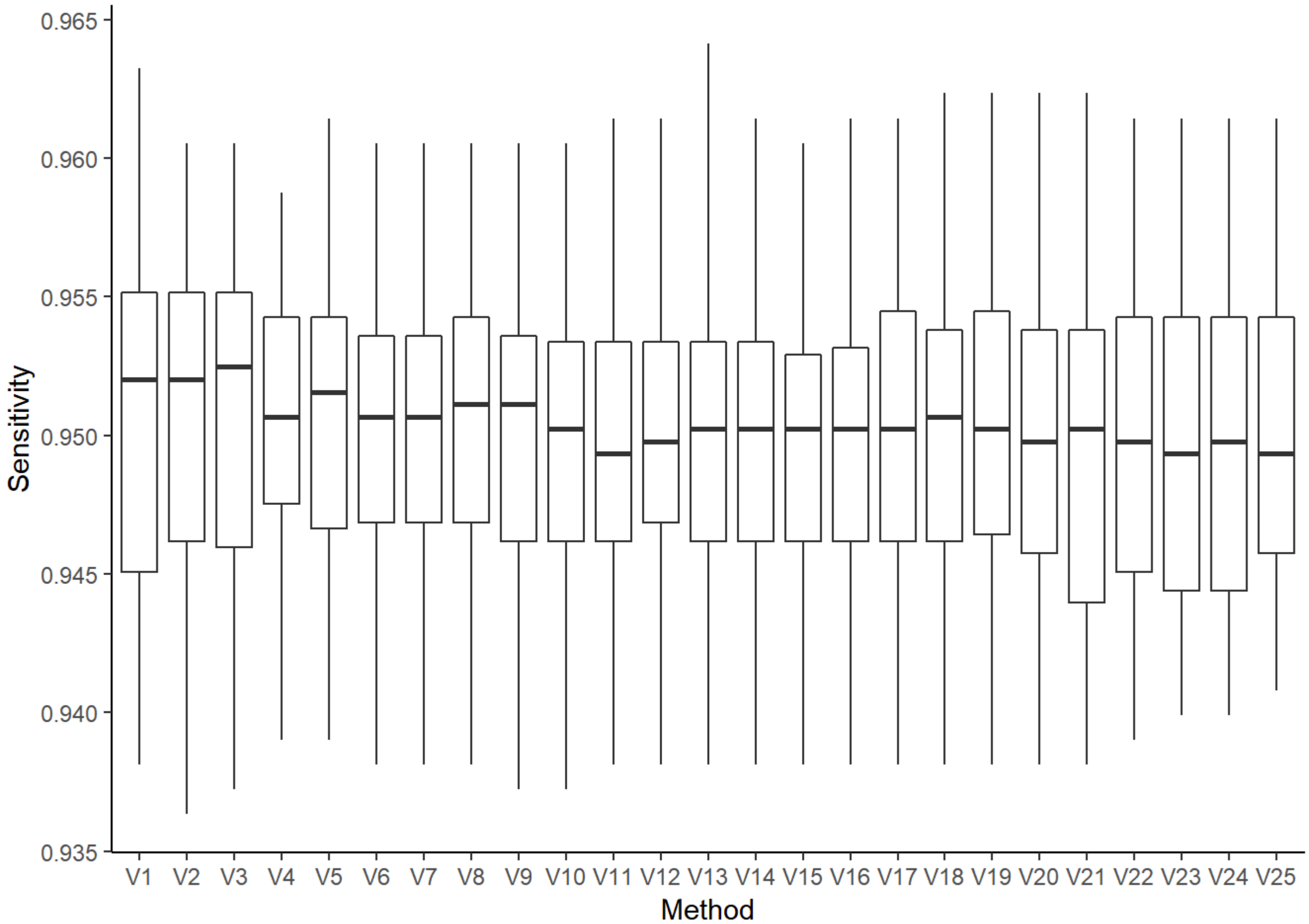

```
# Plot specificity for k-SVM with polynomial kernel
ggplot(spec_mat_melt_poly,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



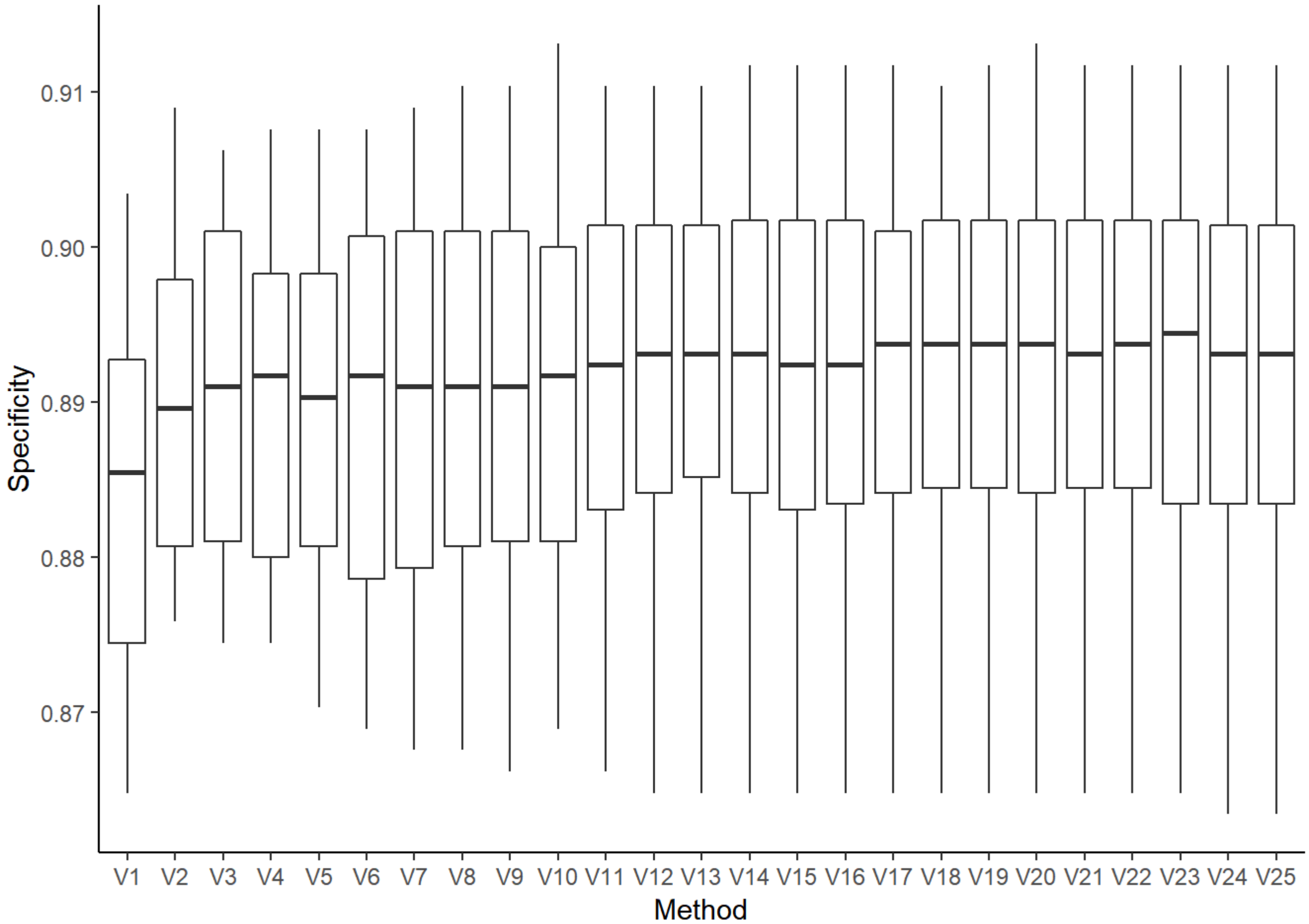

```
# Plot accuracy for k-SVM with linear kernel
ggplot(err_mat_melt_van,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



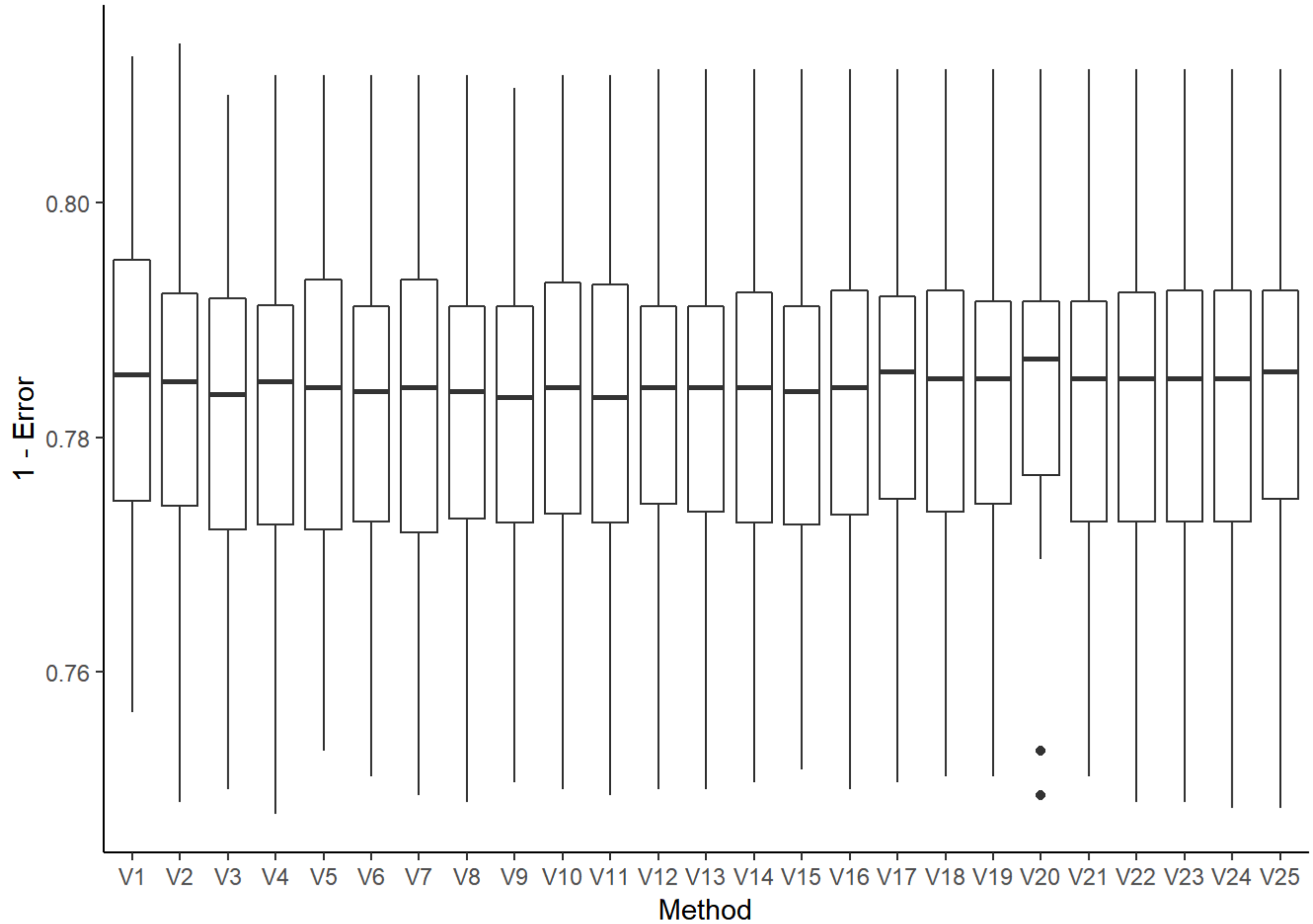

```r
# Plot sensitivity for k-SVM with linear kernel
ggplot(sens_mat_melt_van,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



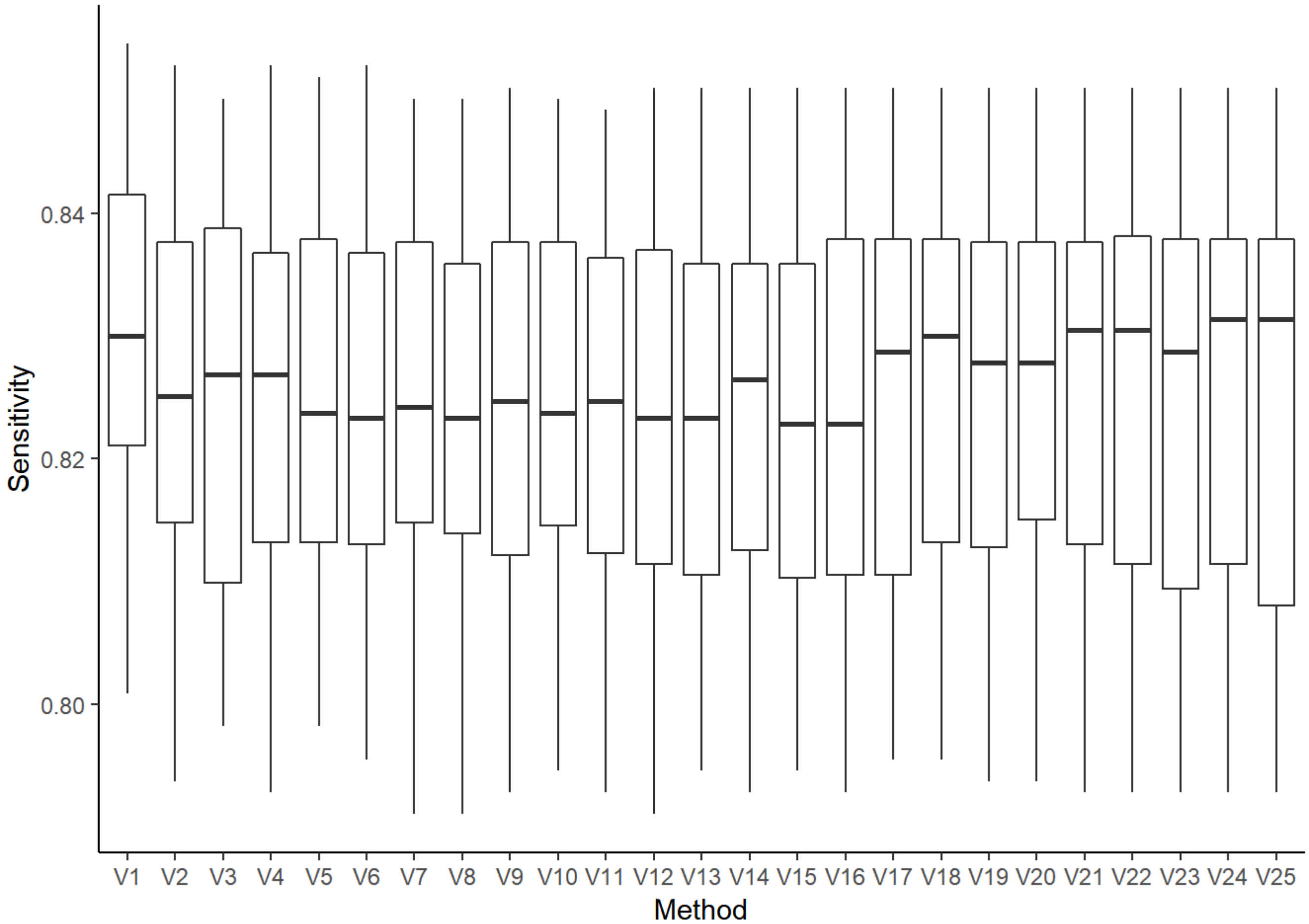

```
# Plot specificity for k-SVM with linear kernel
ggplot(spec_mat_melt_van,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



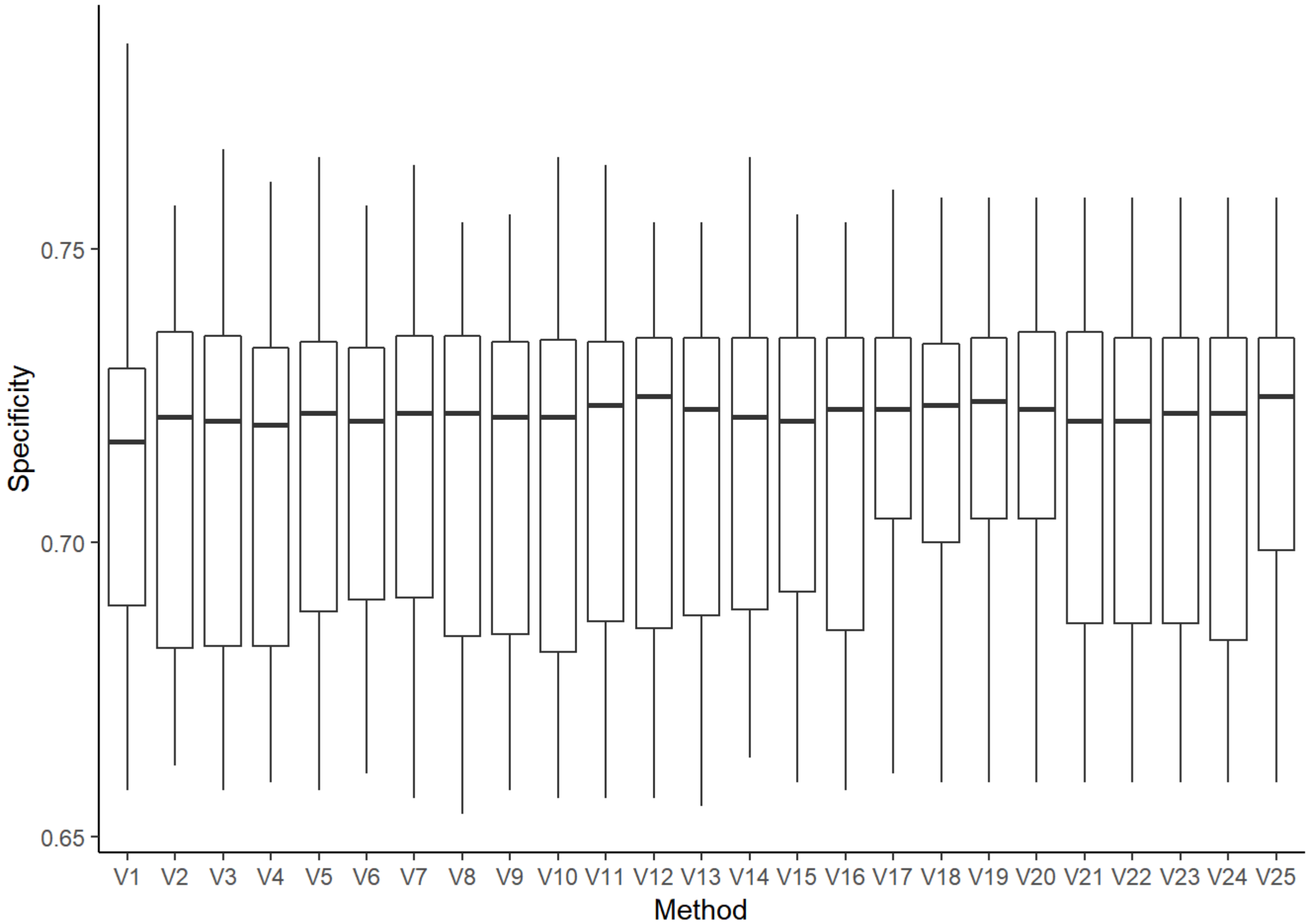

```
# Plot accuracy for k-SVM with hyperbolic tangent kernel
ggplot(err_mat_melt_tanh,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



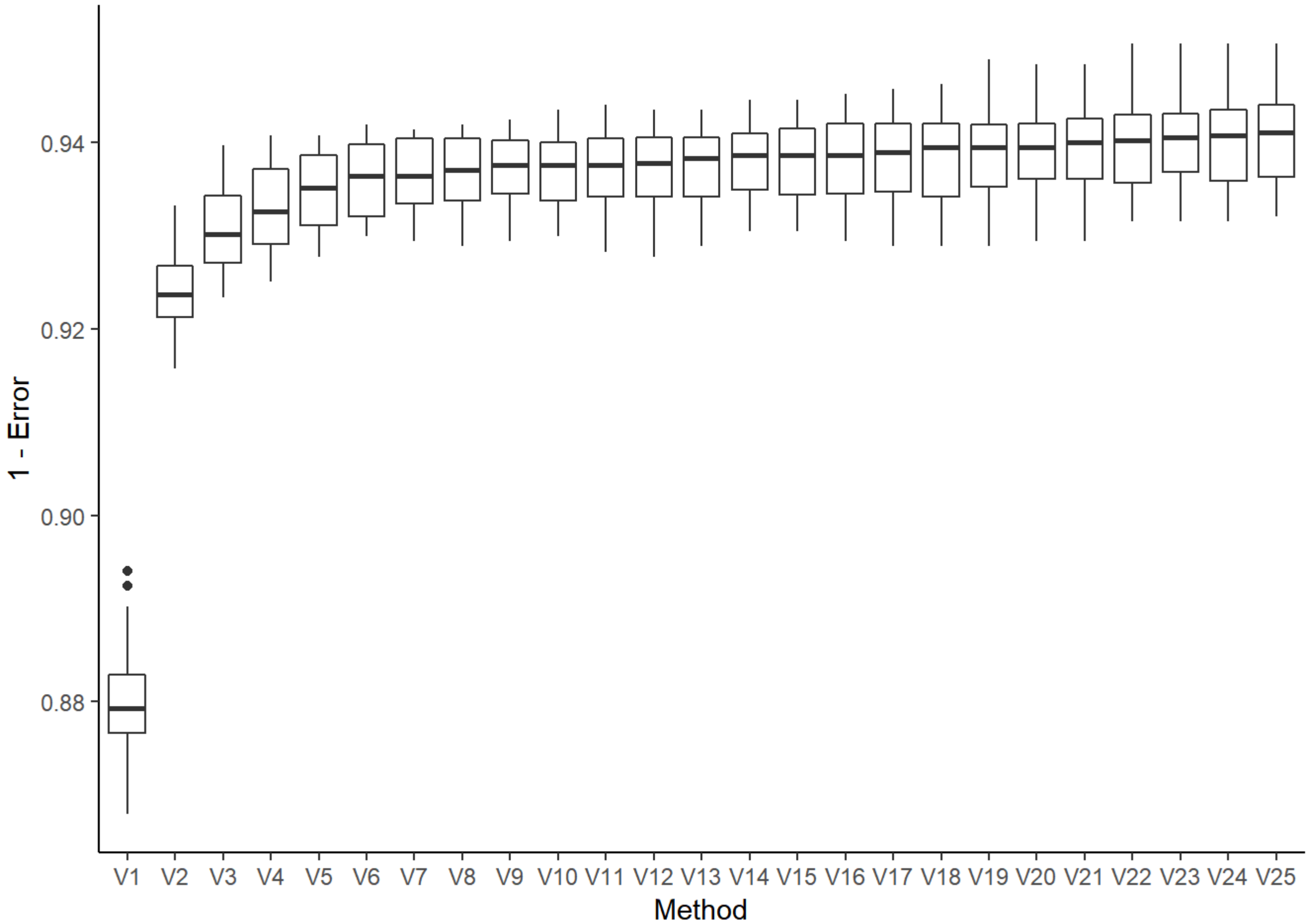

```
# Plot sensitivity for k-SVM with hyperbolic tangent kernel
ggplot(sens_mat_melt_tanh,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



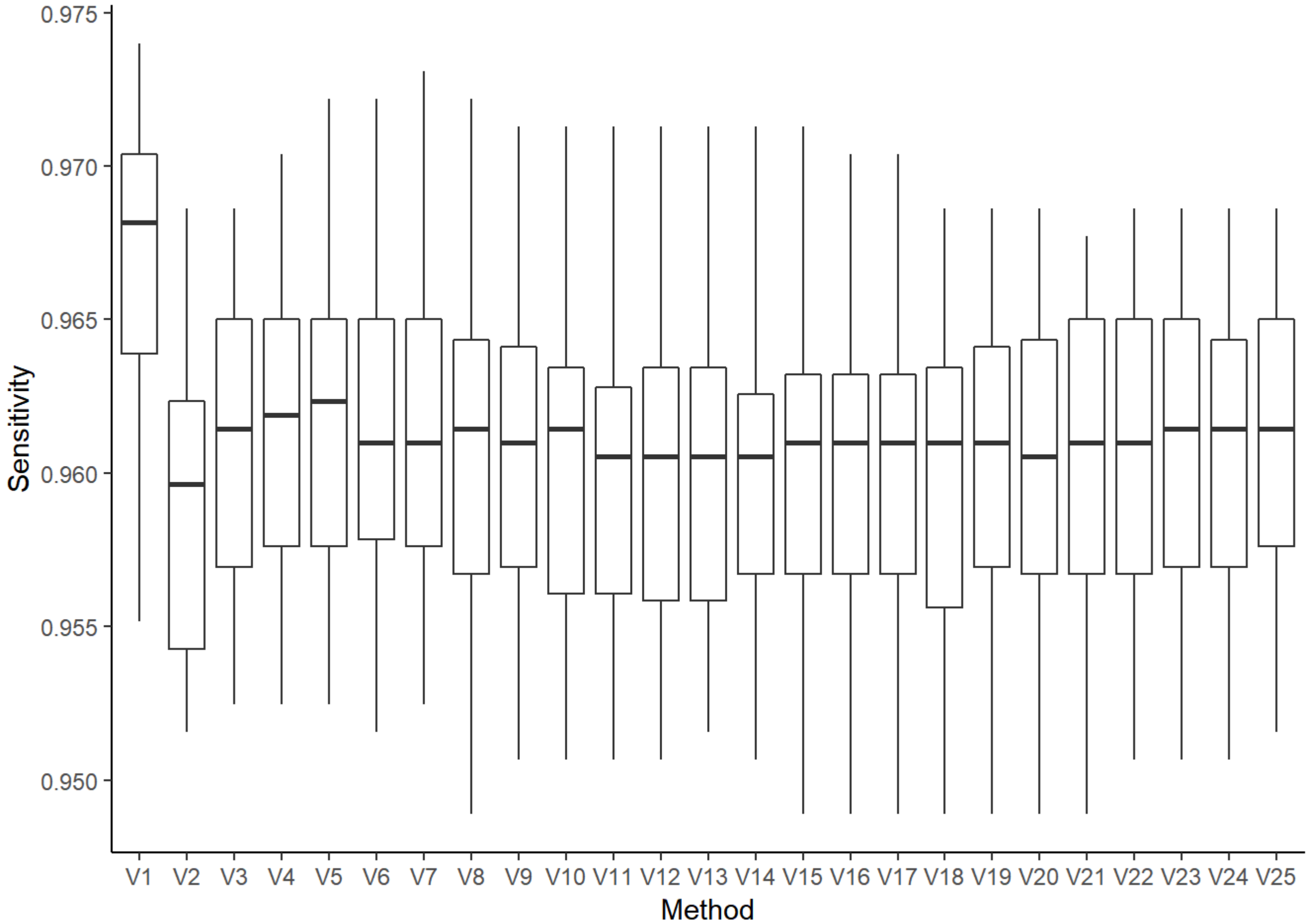

```
# Plot specificity for k-SVM with hyperbolic tangent kernel
ggplot(spec_mat_melt_tanh,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



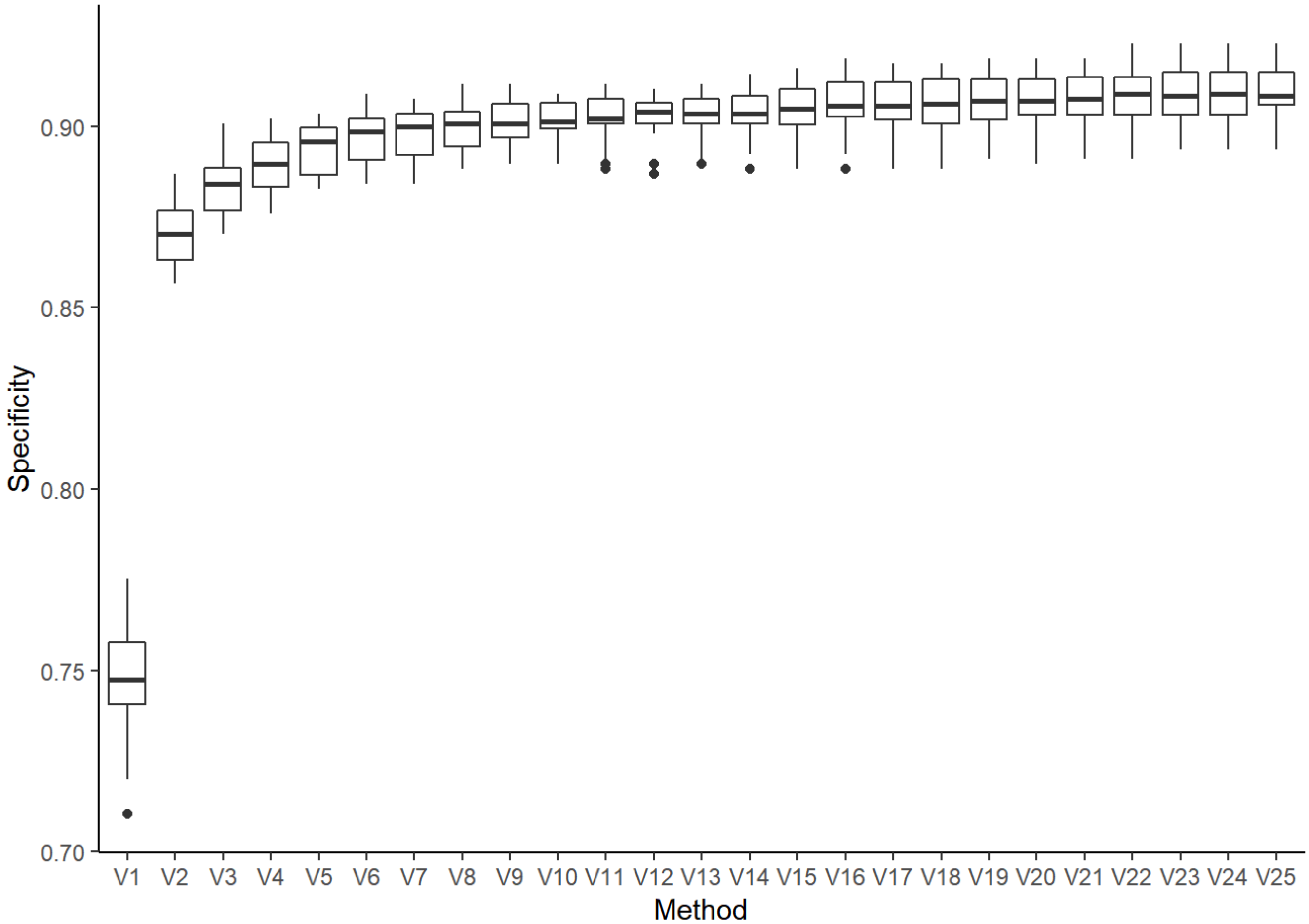

```
# Plot accuracy for k-SVM with Laplacian kernel
ggplot(err_mat_melt_lap,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



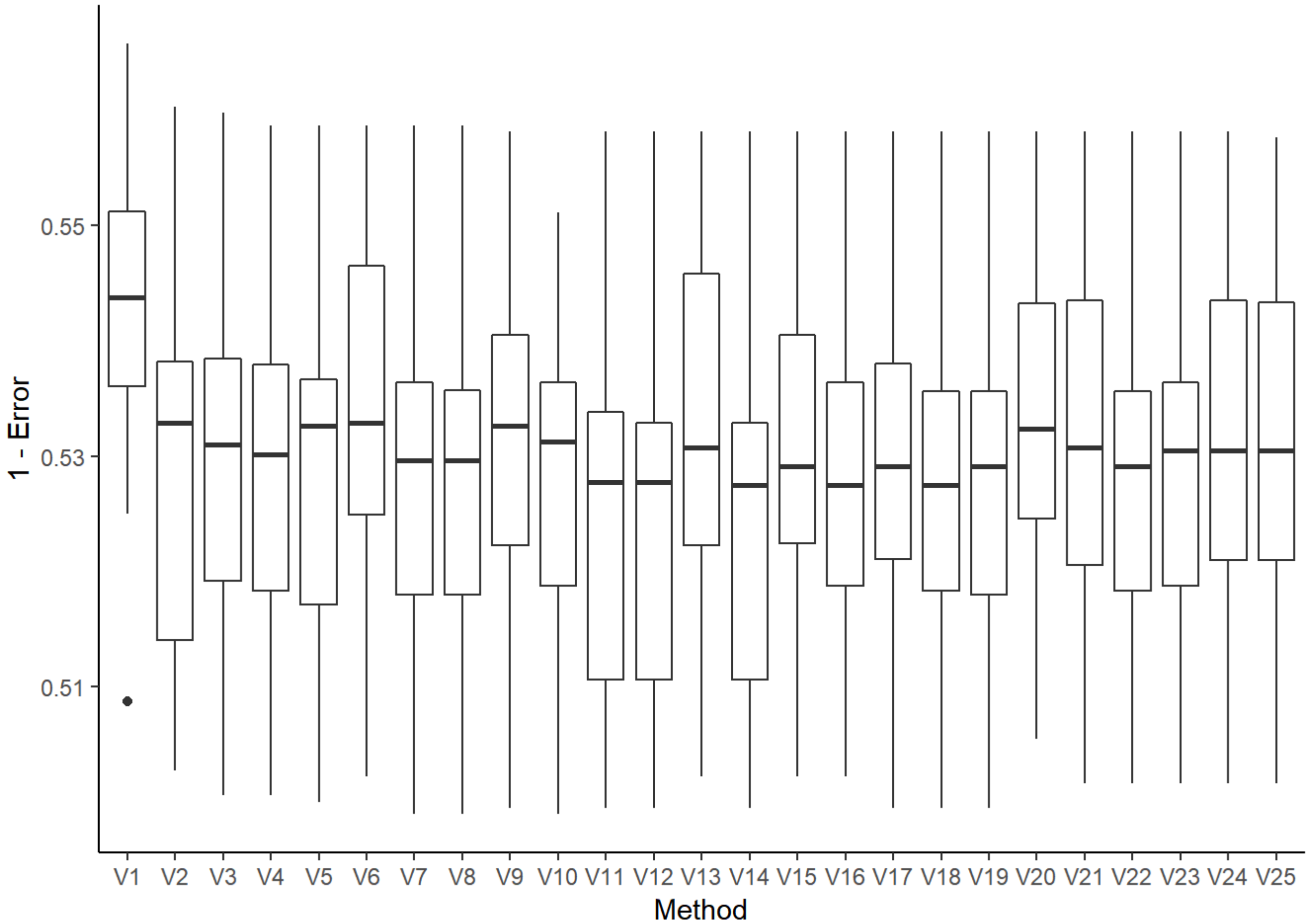

```
# Plot sensitivity for k-SVM with Laplacian kernel
ggplot(sens_mat_melt_lap,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



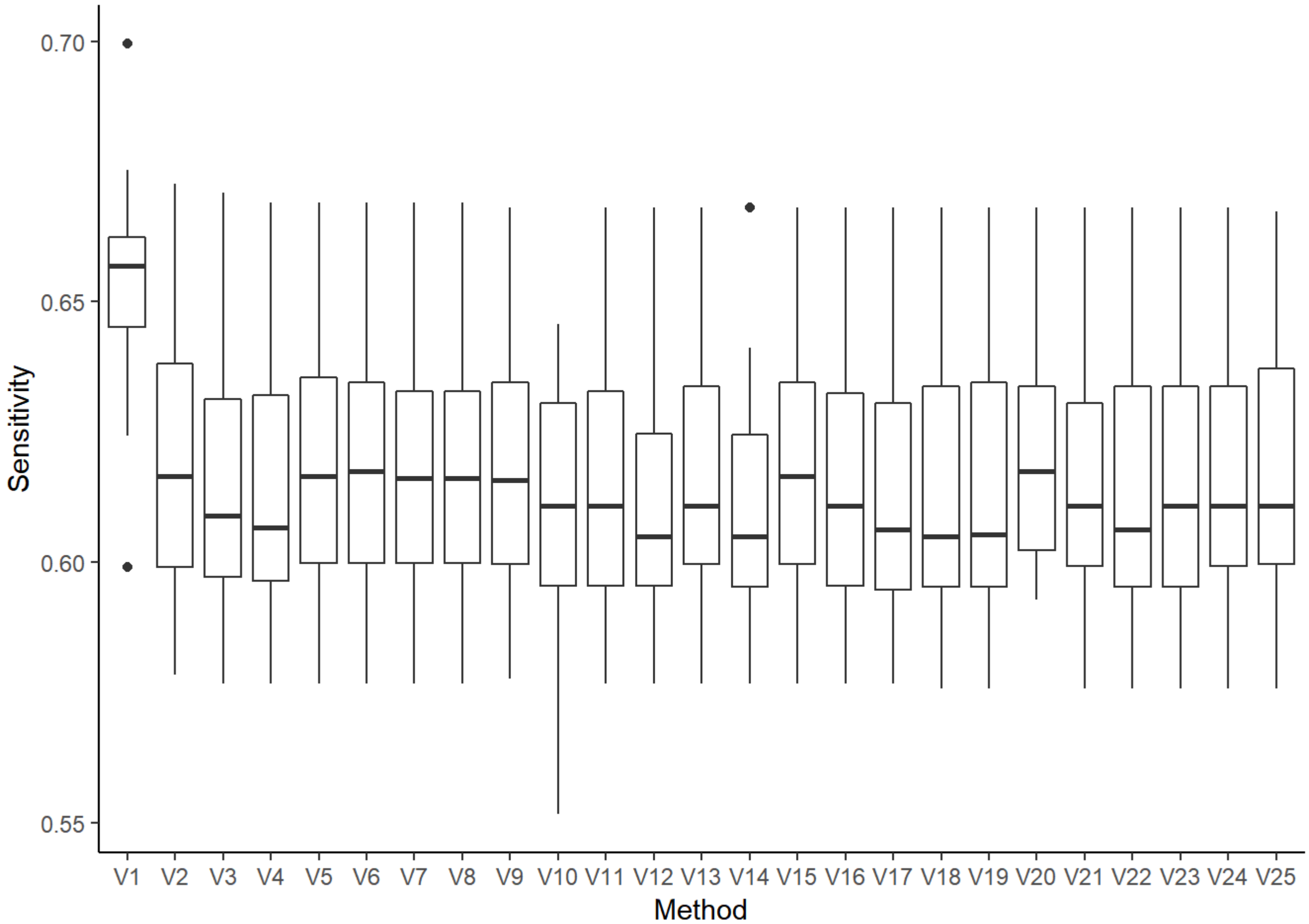

```
# Plot specificity for k-SVM with Laplacian kernel
ggplot(spec_mat_melt_lap,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



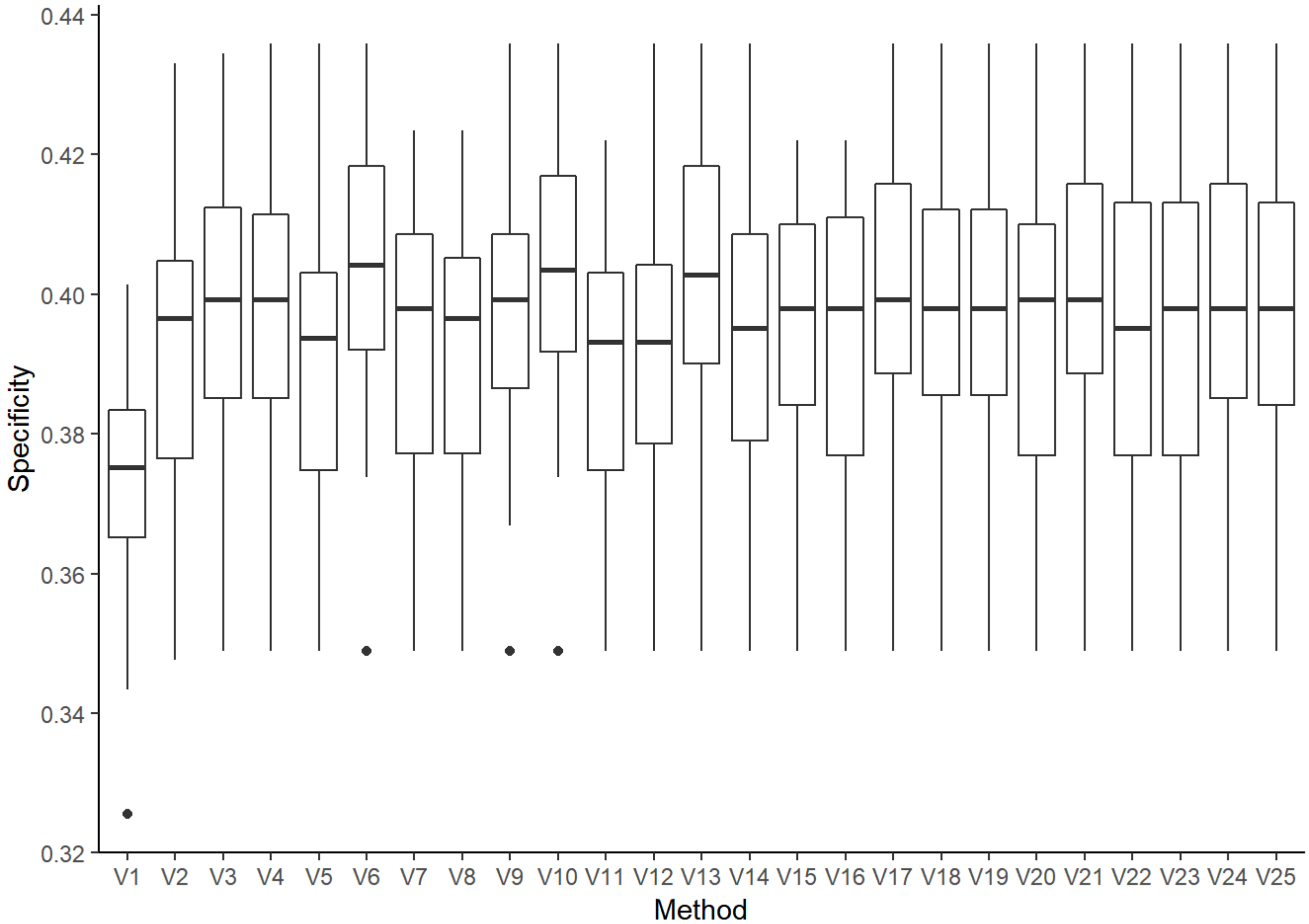

```
# Plot accuracy for k-SVM with Bessel kernel
ggplot(err_mat_melt_bess,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



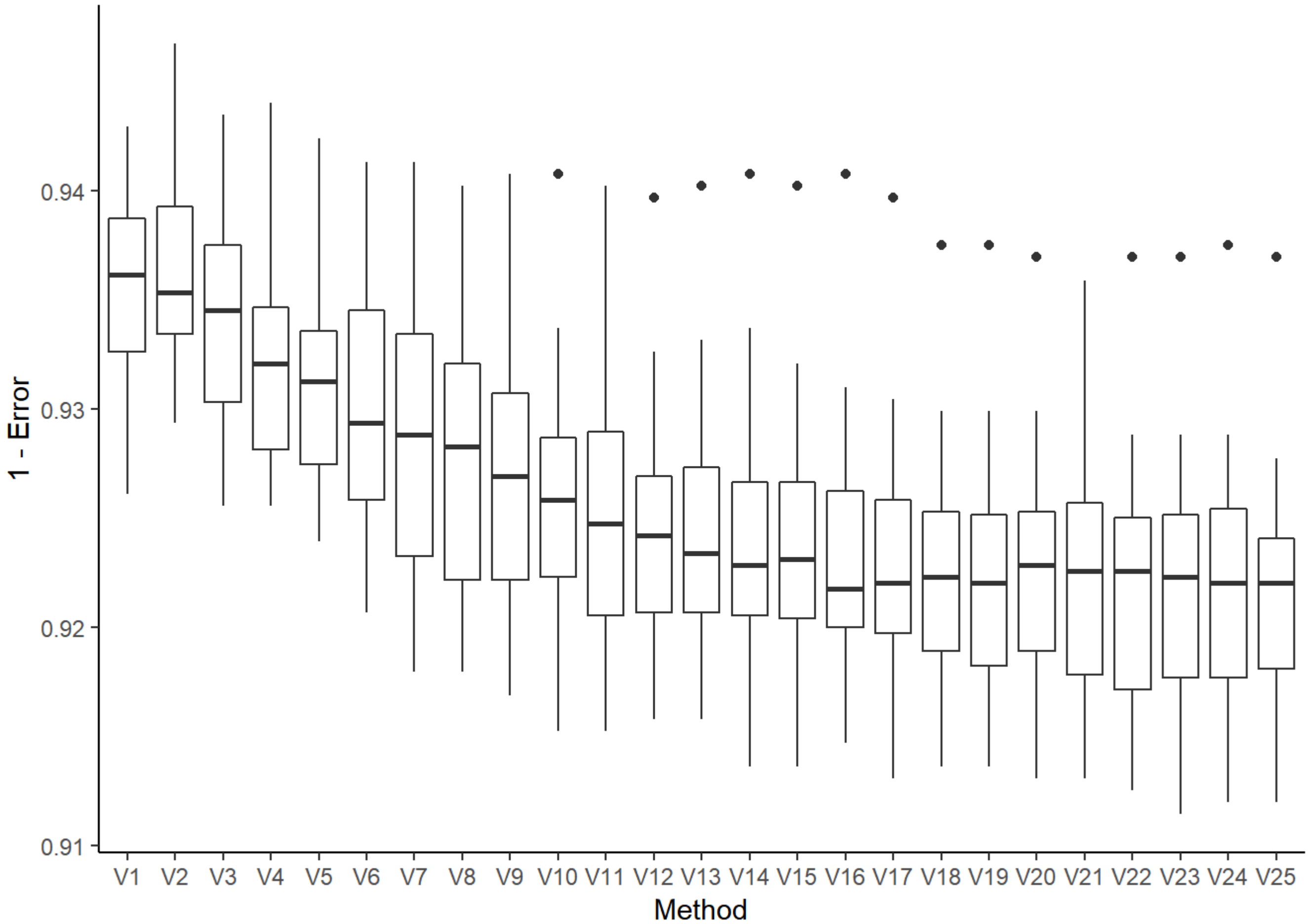

```
# Plot sensitivity for k-SVM with Bessel kernel
ggplot(sens_mat_melt_bess,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



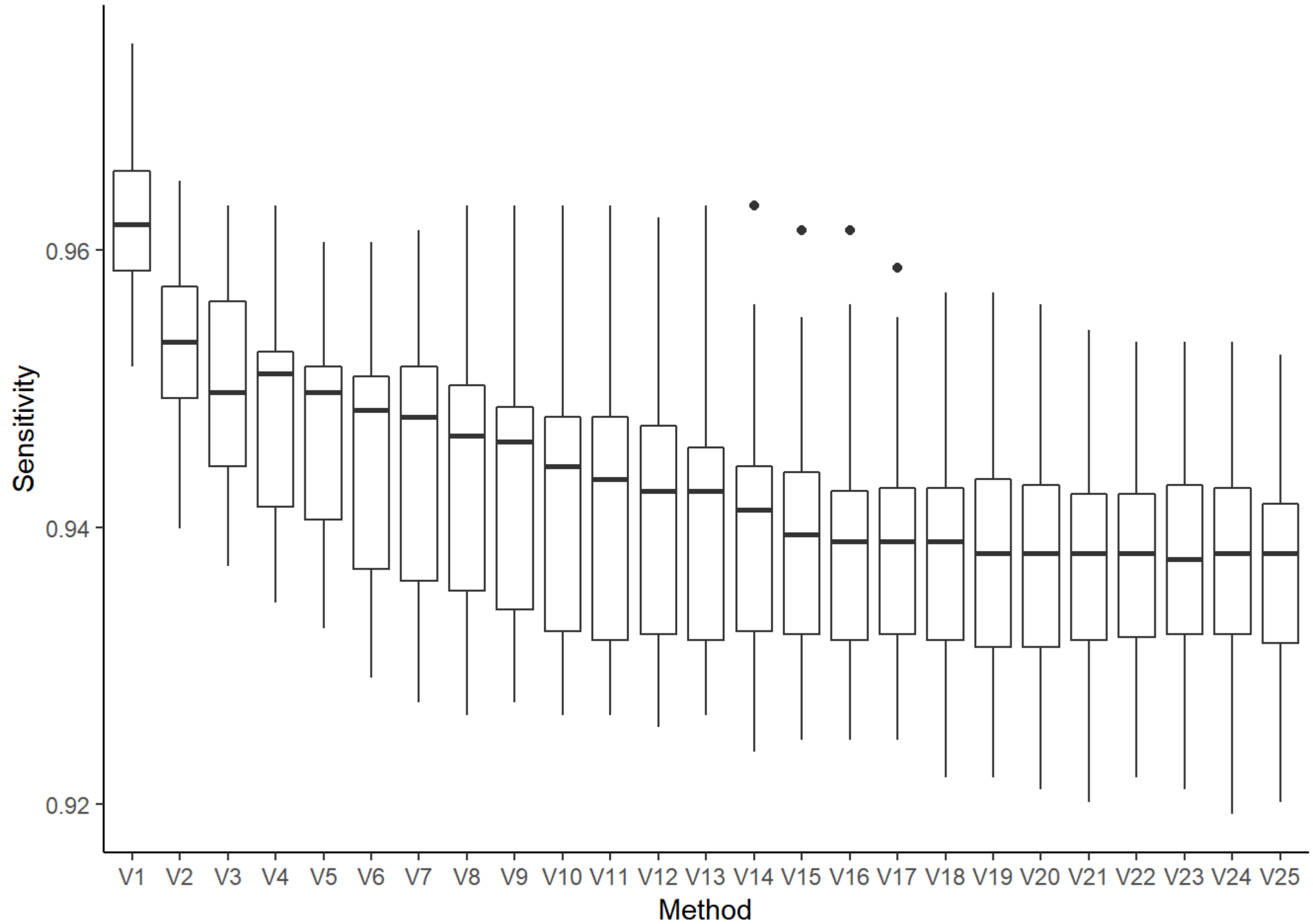

```
# Plot specificity for k-SVM with Bessel kernel
ggplot(spec_mat_melt_bess,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



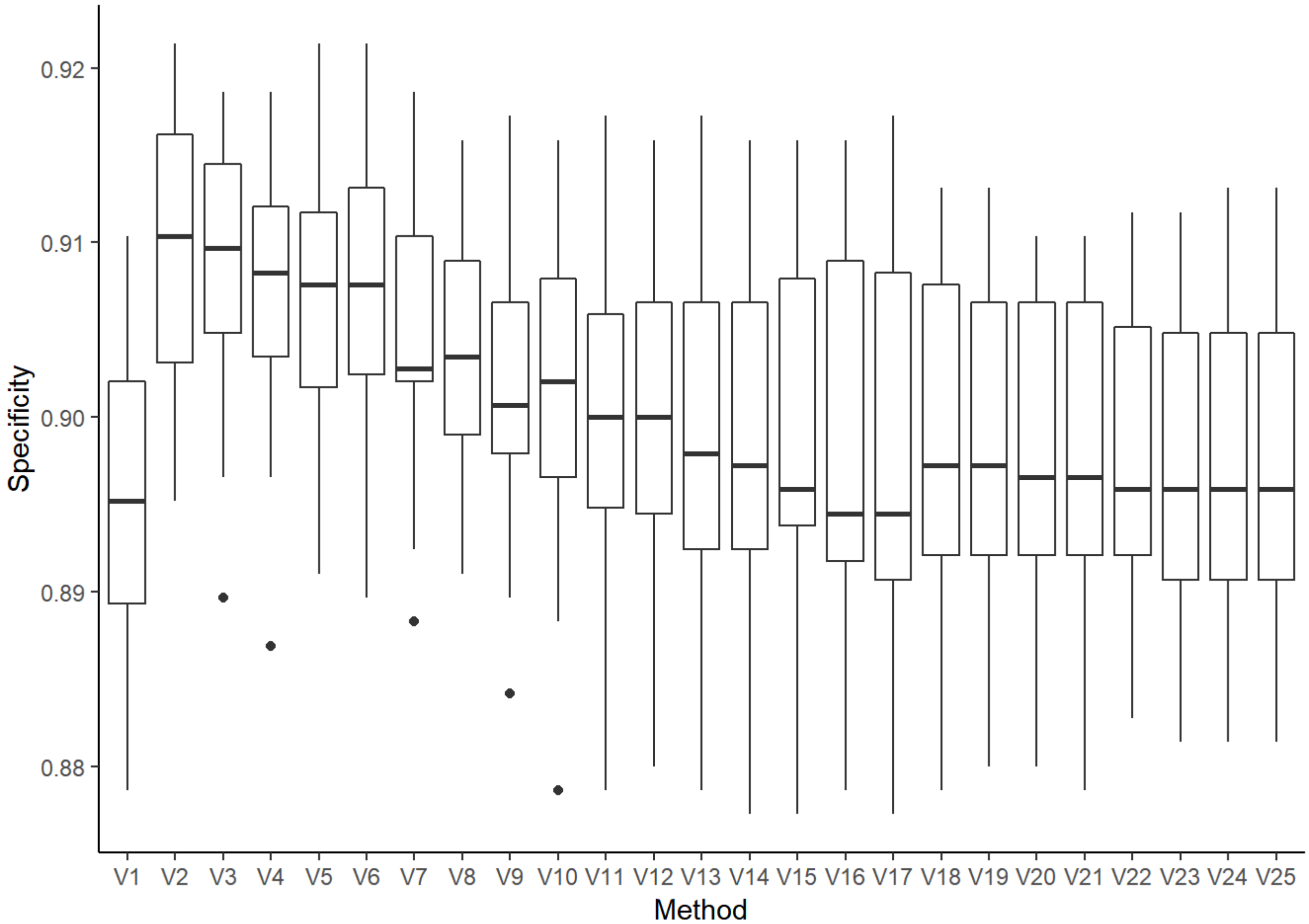

```
# Plot accuracy for k-SVM with ANOVA RBF kernel
ggplot(err_mat_melt_anova,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



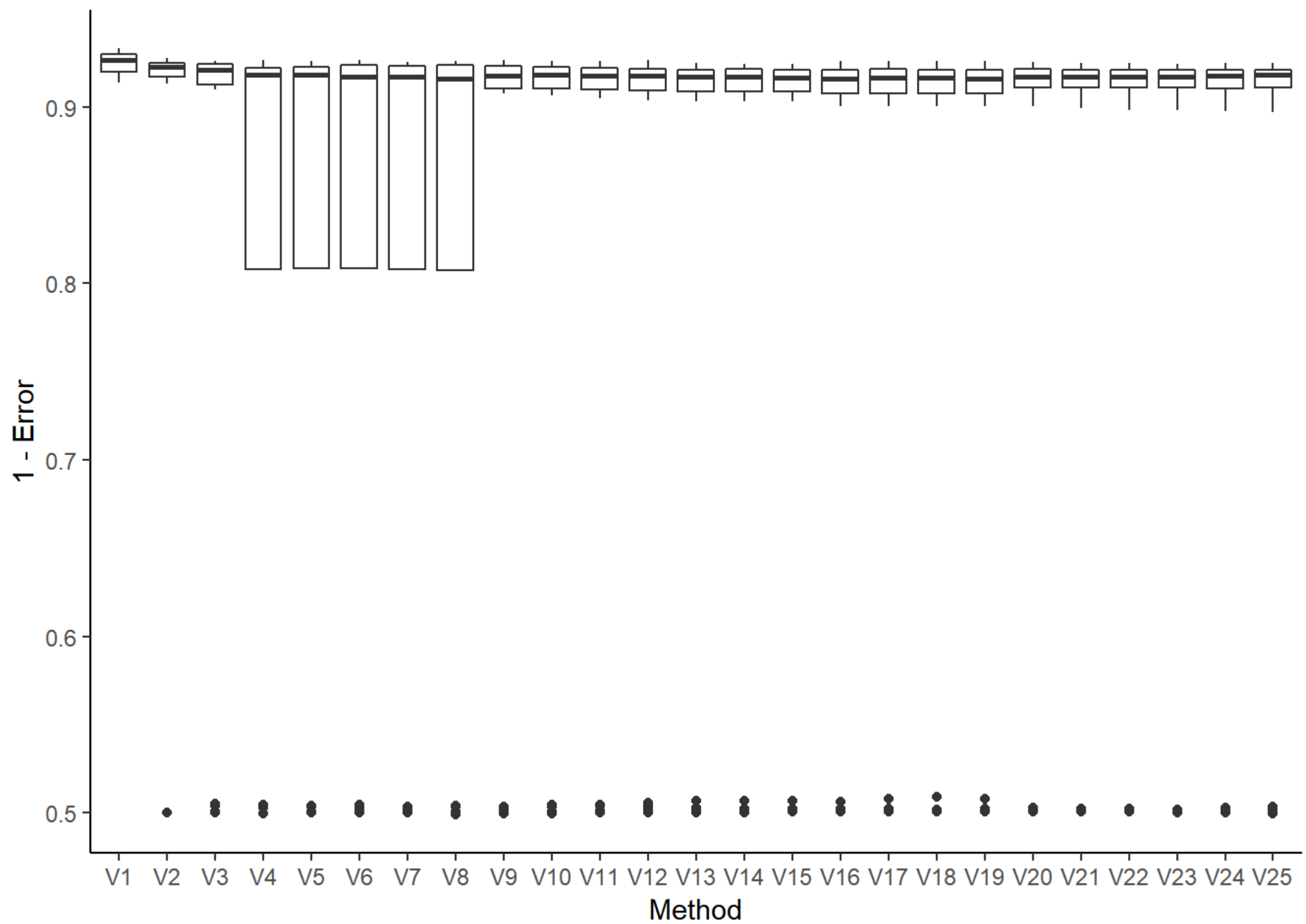

```
# Plot sensitivity for k-SVM with ANOVA RBF kernel
ggplot(sens_mat_melt_anova,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```



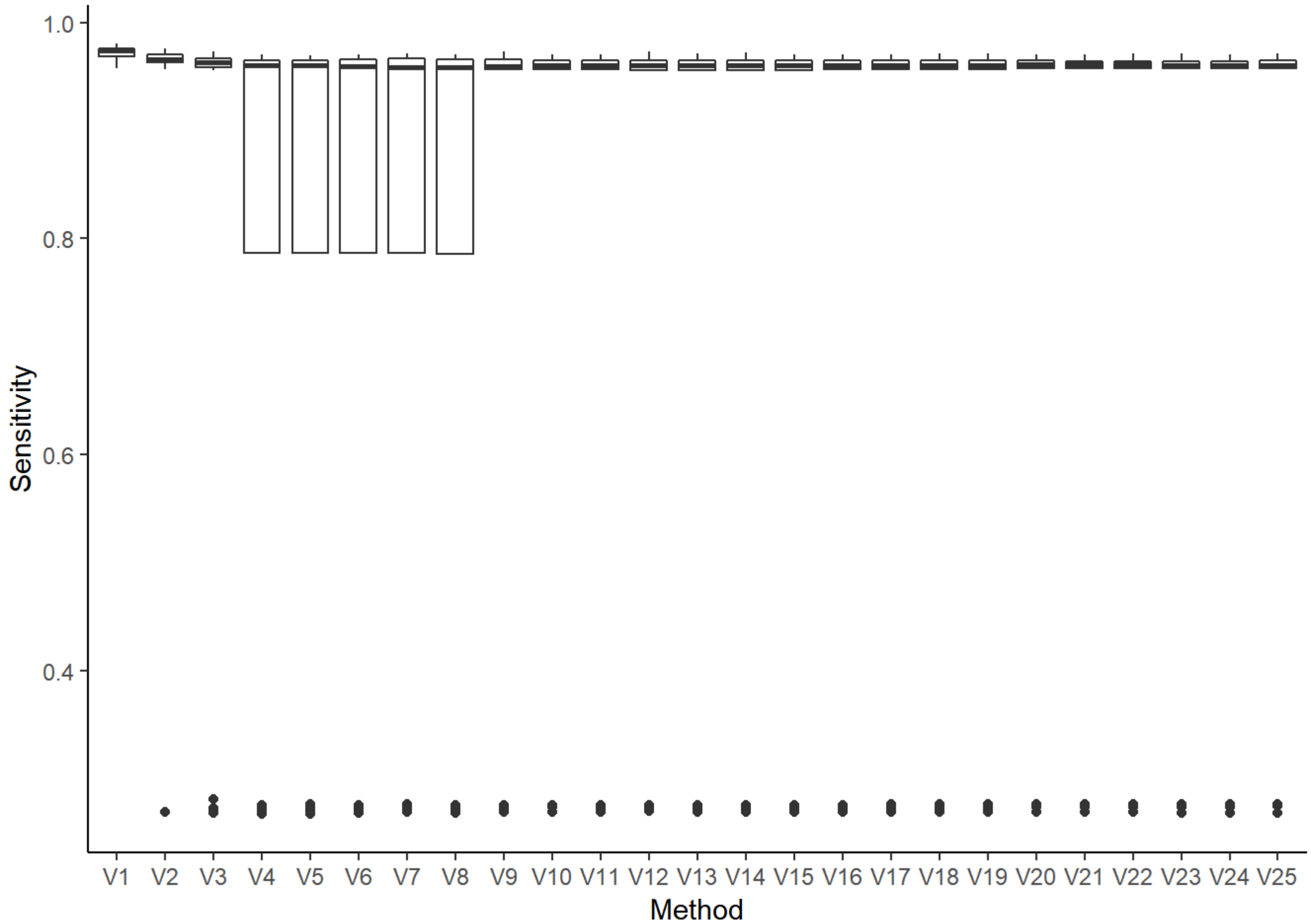

```
# Plot specificity for k-SVM with ANOVA RBF kernel
ggplot(spec_mat_melt_anova,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



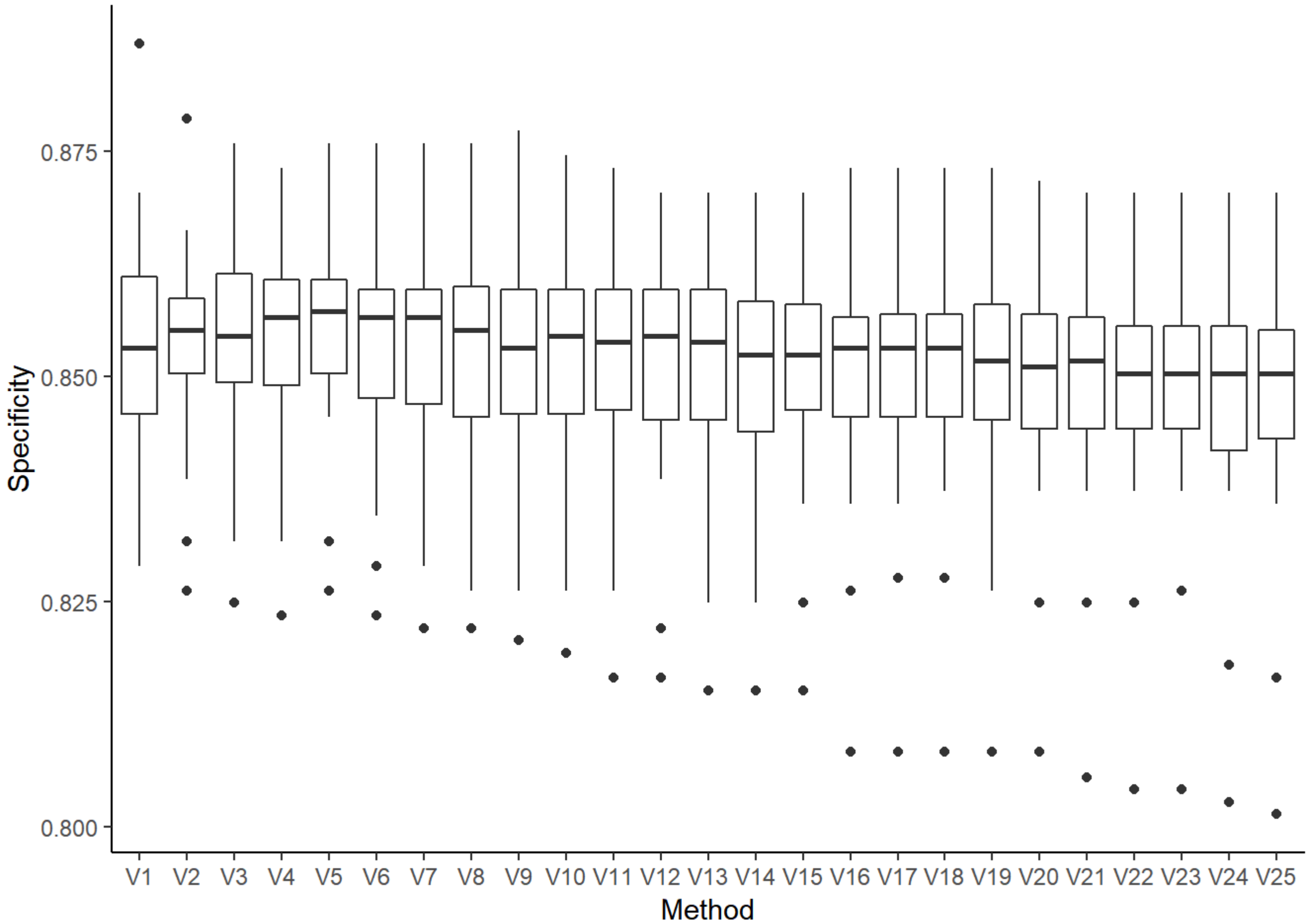

```
# Plot accuracy for k-SVM with spline kernel
ggplot(err_mat_melt_spline,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```




```
# Plot sensitivity for k-SVM with spline kernel
ggplot(sens_mat_melt_spline,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```




```r
# Plot specificity for k-SVM with spline kernel
ggplot(spec_mat_melt_spline,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



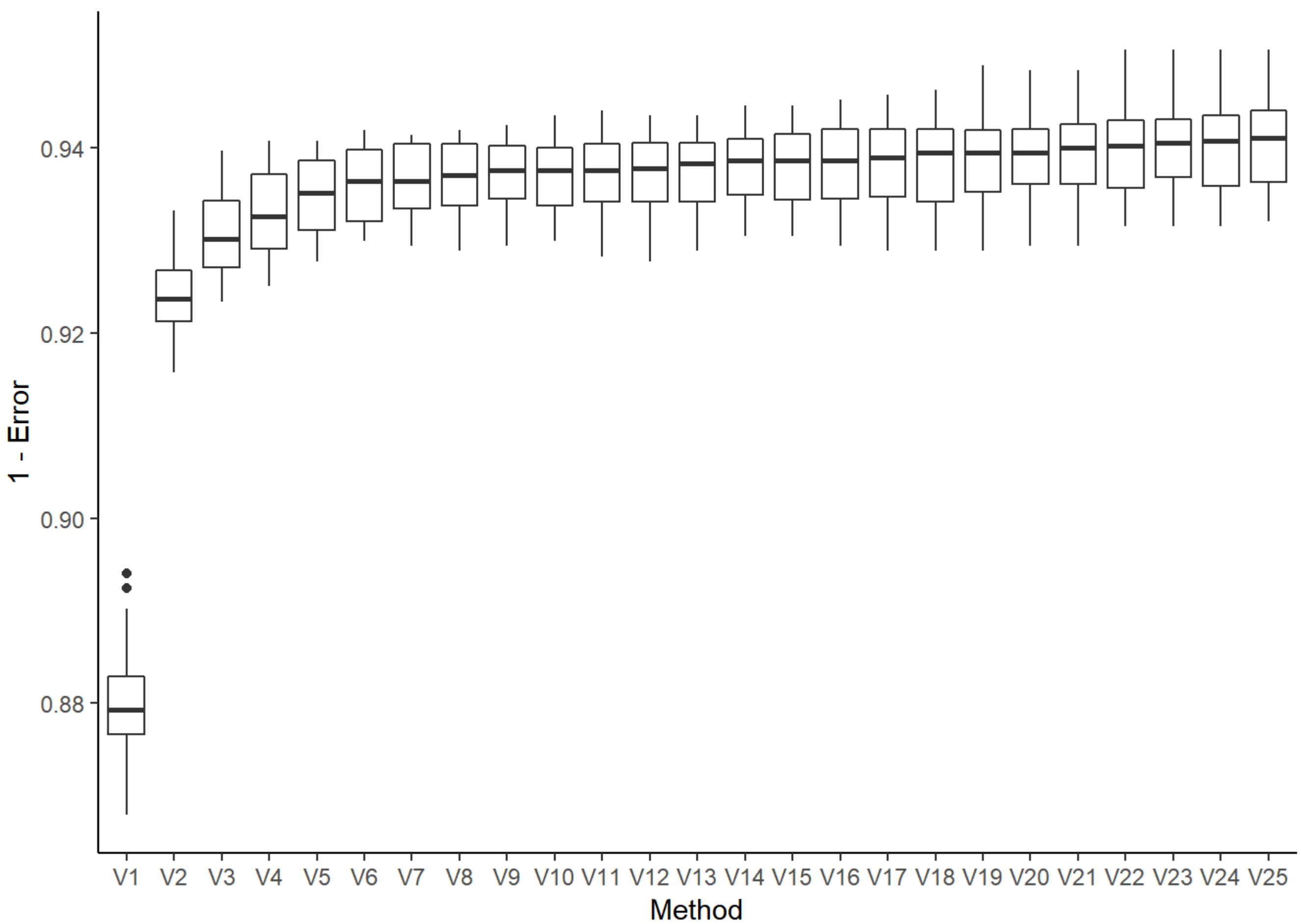
Higher values of C need to be tested with a Laplacian kernel.

More KSVM tuning

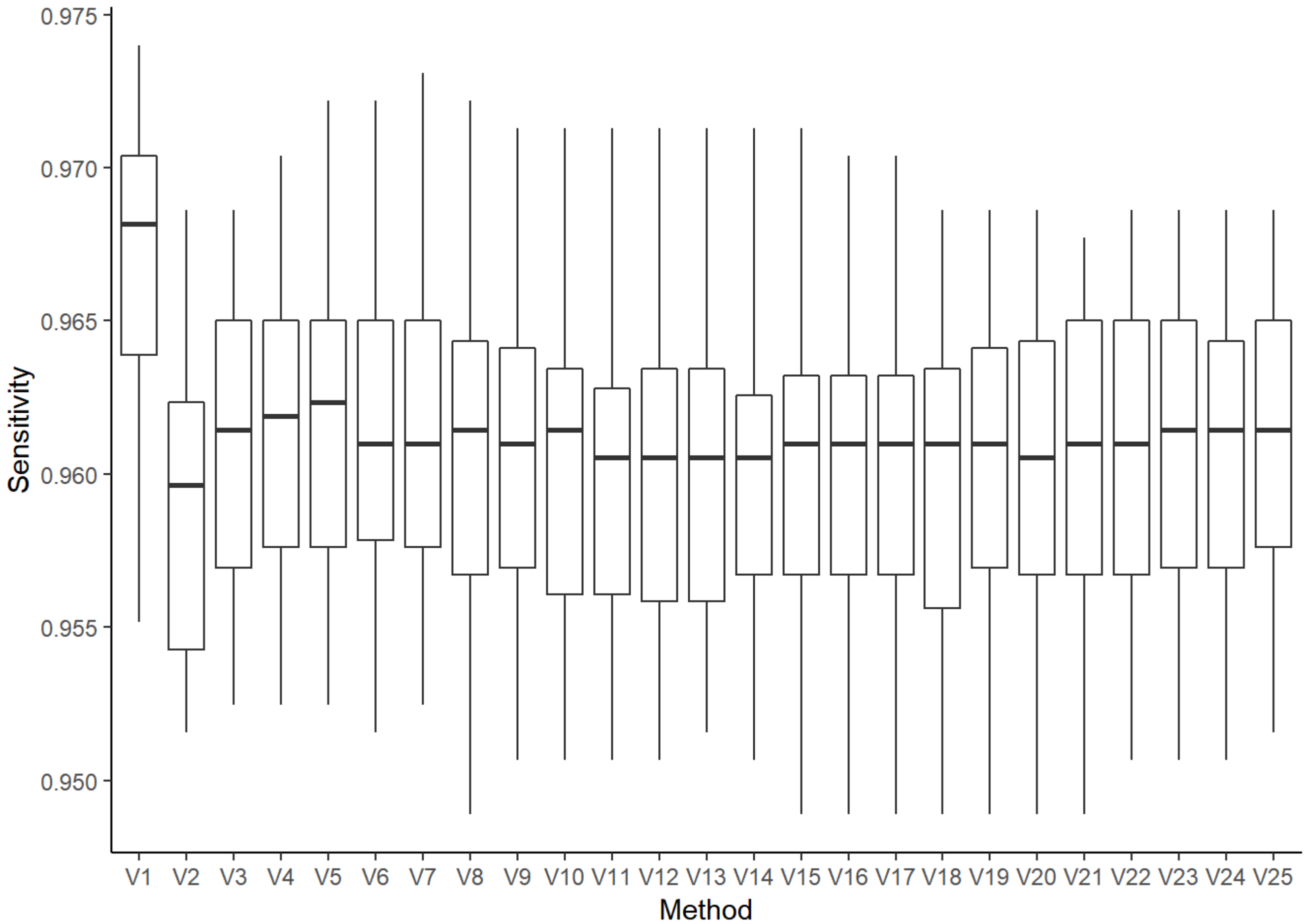
```r
R = 20 # set the number of replications

set.seed(1)

# Set number of values of C to test
n_c <- 25

# Create sequence of values of C to test
v_c = seq(15, 2^7, length=n_c)

# create the error matrix to store values
err_mat_lap2 = matrix(0,
                     ncol=n_c,
                     nrow=R)

# create sensitivity matrix to store values
sens_mat_lap2 = matrix(0,
                      ncol=n_c,
                      nrow=R)

# create specificity matrix to store values
spec_mat_lap2 = matrix(0,
                      ncol=n_c,
                      nrow=R)

# Loop through the repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through values of C
  for(n in 1:n_c) {
    # Run k-SVM with Laplacian kernel, predict, and calculate metrics
    mod_lap2 <- ksvm(spam~.,
                     spam_train,
                     cross=0,
                     C=v_c[n],
                     kernel = "laplacedot",
                     type='C-svc',
                     metric = "ROC")
    yhat_lap2 = predict(mod_lap2, spam_test[,-58])
    err_mat_lap2[r,n] = mean(yhat_lap2!=spam_test[,58])
    cm_lap2 <- confusionMatrix(yhat_lap2, spam_test[,58], positive = "No")
    sens_mat_lap2[r,n] = cm_lap2$byClass["Sensitivity"]
    spec_mat_lap2[r,n] = cm_lap2$byClass["Specificity"]
  }
```

```
  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_lap2 = melt(as.data.frame(err_mat_lap2))
colnames(err_mat_melt_lap2) = c('Method','Error')

sens_mat_melt_lap2 = melt(as.data.frame(sens_mat_lap2))
colnames(sens_mat_melt_lap2) = c('Method','Sensitivity')

spec_mat_melt_lap2 = melt(as.data.frame(spec_mat_lap2))
colnames(spec_mat_melt_lap2) = c('Method','Specificity')
```

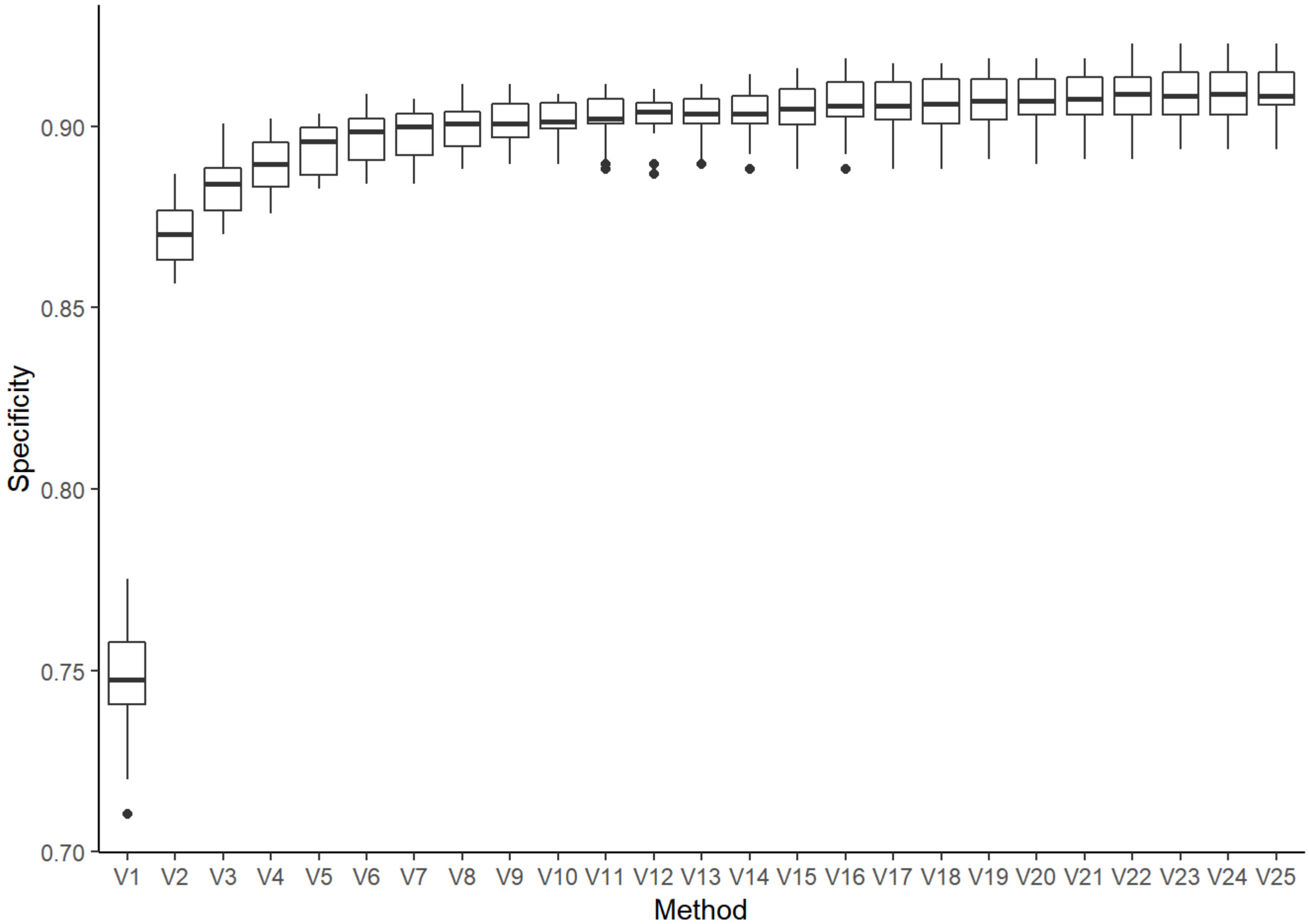
```
# Plot accuracy for k-SVM with Laplacian kernal
ggplot(err_mat_melt_lap,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```




```
# Plot sensitivity for k-SVM with Laplacian kernel
ggplot(sens_mat_melt_lap,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```




```
# Plot specificity for k-SVM with Laplacian kernel
ggplot(spec_mat_melt_lap,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```

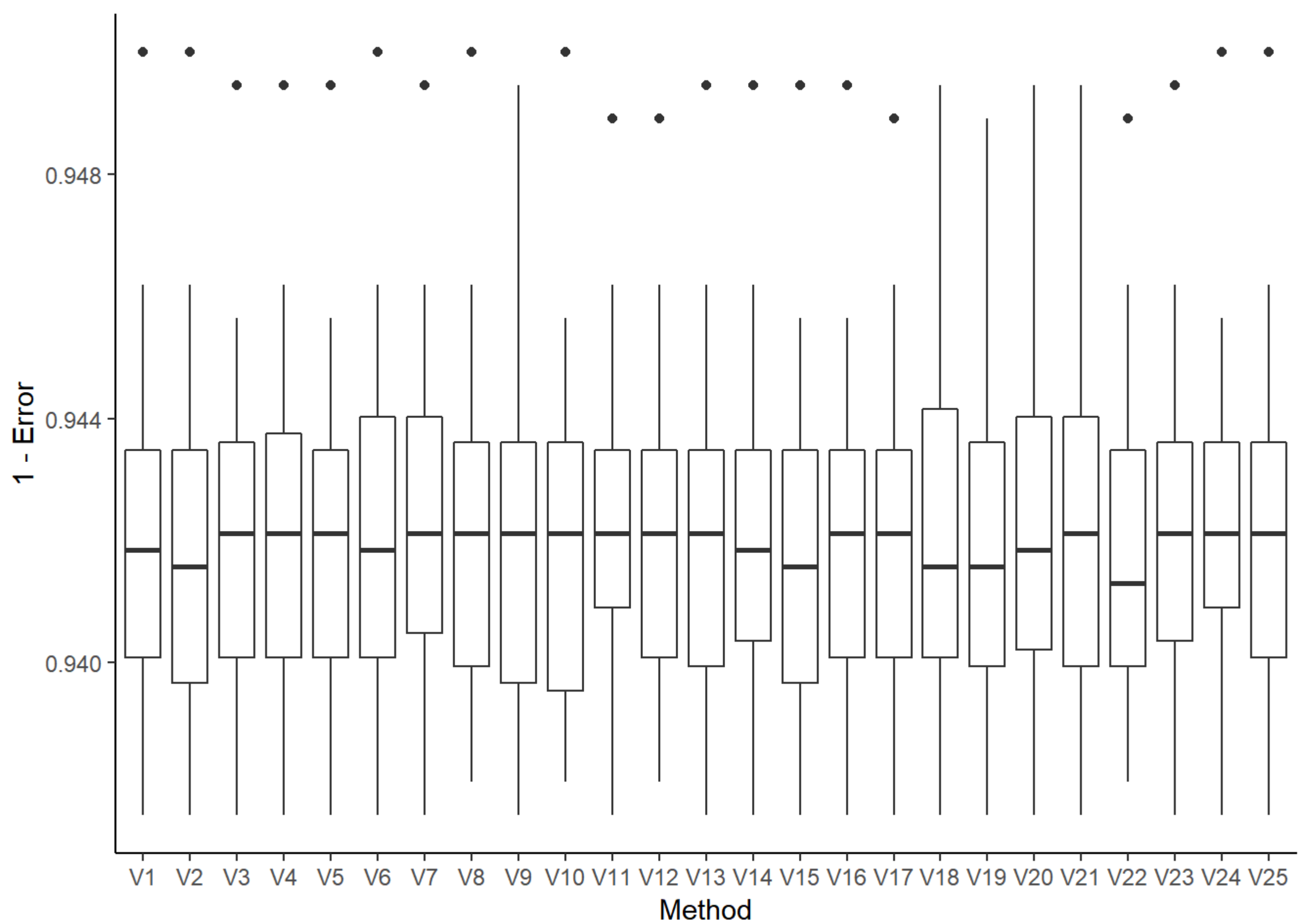


We can also tune the sigma hyperparameter for k-SVM with a Laplacian kernel.

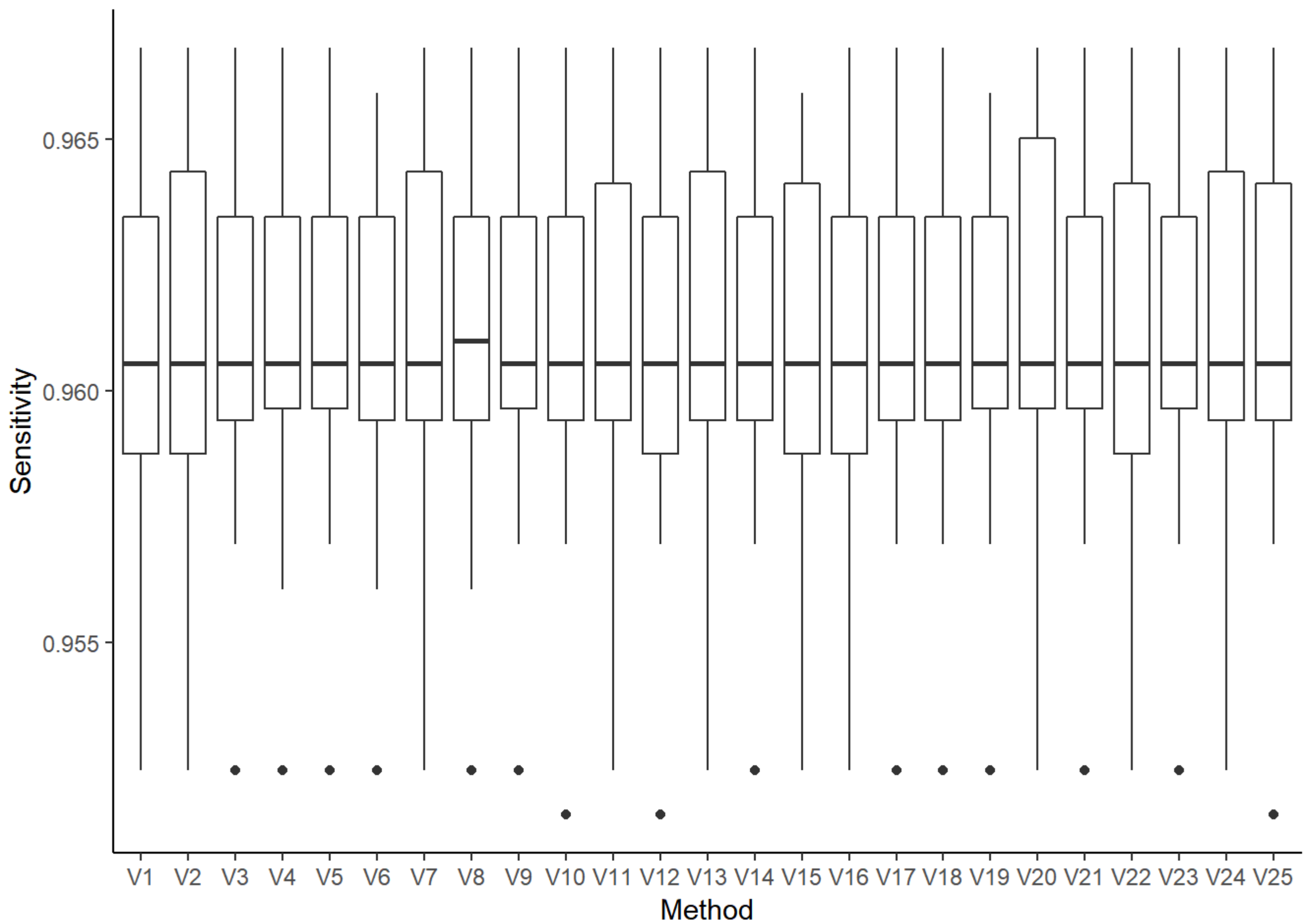
```r
R = 20 # set the number of replications

set.seed(1)

# Set number of values of sigma to test
n_sig <- 25

# Create sequence of values of sigma to test
v_sig = seq(0.1, 10, length=n_sig)

# create the error matrix to store values
err_mat_lap_sig = matrix(0,
                     ncol=n_sig,
                     nrow=R)

# create sensitivity matrix to store values
sens_mat_lap_sig = matrix(0,
                      ncol=n_sig,
                      nrow=R)

# create specificity matrix to store values
spec_mat_lap_sig = matrix(0,
                      ncol=n_sig,
                      nrow=R)

# Loop through the repetitions
for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Loop through the values of sigma
  for(n in 1:n_sig) {
    # Run k-SVM model with Laplacian kernel, predict, and calculate metrics
    mod_lap <- ksvm(spam~.,
                  spam_train,
                  cross=0,
                  C=15,
                  kernel = "laplacedot",
                  sigma = v_sig[n_sig],
                  type='C-svc',
                  metric = "ROC")
    yhat_lap_sig = predict(mod_lap, spam_test[,-58])
    err_mat_lap_sig[r,n] = mean(yhat_lap_sig!=spam_test[,58])
    cm_lap_sig <- confusionMatrix(yhat_lap_sig, spam_test[,58], positive = "No")
    sens_mat_lap_sig[r,n] = cm_lap_sig$byClass["Sensitivity"]
    spec_mat_lap_sig[r,n] = cm_lap_sig$byClass["Specificity"]
```

```
  }


  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}

# Melt output to prepare for plotting
err_mat_melt_lap_sig = melt(as.data.frame(err_mat_lap_sig))
colnames(err_mat_melt_lap_sig) = c('Method','Error')

sens_mat_melt_lap_sig = melt(as.data.frame(sens_mat_lap_sig))
colnames(sens_mat_melt_lap_sig) = c('Method','Sensitivity')

spec_mat_melt_lap_sig = melt(as.data.frame(spec_mat_lap_sig))
colnames(spec_mat_melt_lap_sig) = c('Method','Specificity')
```

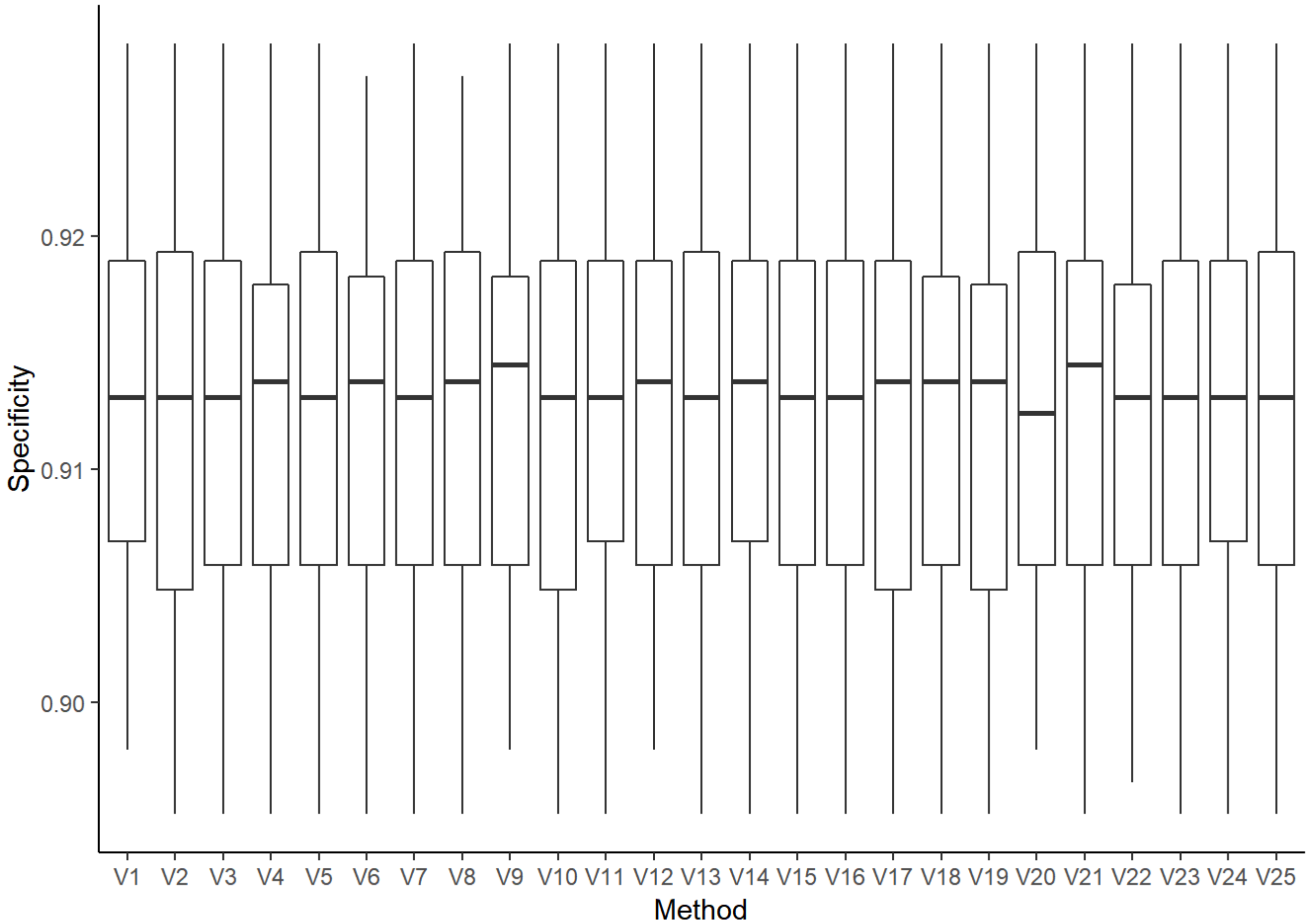
```
# Plot accuracy for k-SVM with Laplacian kernel
ggplot(err_mat_melt_lap_sig,mapping=aes(x=Method,y=1-Error))+
  geom_boxplot() +
  theme_classic()
```



```r
# Plot sensitivity for k-SVM with Laplacian kernel
ggplot(sens_mat_melt_lap_sig,mapping=aes(x=Method,y=Sensitivity))+
  geom_boxplot() +
  theme_classic()
```




```r
# Plot specificity for k-SVM with Laplacian kernel
ggplot(spec_mat_melt_lap_sig,mapping=aes(x=Method,y=Specificity))+
  geom_boxplot() +
  theme_classic()
```



Changing sigma has minimal effect, so none of the k-SVM models matches the performance of the optimized random forests.

Finally, we need to output confusion matrices for the base random forests and optimized random forests for comparison.

Rerun random forest: best optimized vs. base and get final metric outputs

```r
R = 50 # set the number of replications

# set up train control to do CV
#tunegrid = expand.grid(mtry=c(6:10))

fitControl = trainControl(method = "cv",
                          number = 5,
                          returnData = TRUE,
                          returnResamp = "final",
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE)

# create the error matrix to store values
err_mat = matrix(0, ncol=2, nrow=R)

# create sensitivity matrix to store values
sens_mat = matrix(0, ncol=2, nrow=R)

# create specificity matrix to store values
spec_mat = matrix(0, ncol=2, nrow=R)

# create list to store confusion matrices
cm_reg_list = vector("list", R)
cm_rf_list = vector("list", R)
```

```r
set.seed(1)

for (r in 1:R){

  # training test split
  id = holdout(spambase$spam,
               ratio=.6,
               mode='stratified')

  # Create training and test sets
  spam_train = spambase[id$tr,]
  spam_test = spambase[id$ts,]

  # Run base random forest model, predict, and calculate metrics and
  # confusion matrices
  mod_reg = randomForest(spam ~ .,
                         spam_train,
                         ntree = 100,
                         trControl = fitControl,
                         metric = "ROC")


  yhat_reg = predict(mod_reg, spam_test[,-58])

  err_mat[r,1] = mean(yhat_reg!=spam_test[,58])

  cm_reg <- confusionMatrix(yhat_reg, spam_test[,58], positive = "No")
  cm_reg_list[[r]] = cm_reg

  sens_mat[r,1] = cm_reg$byClass["Sensitivity"]

  spec_mat[r,1] = cm_reg$byClass["Specificity"]

  # Set mtry to optimal value
  num_var <- c(6)

  # Run optimized random forest model, predict, and calculate metrics and
  # confusion matrices
  mod_rf = train(spam ~ .,
                 spam_train,
                 trControl = fitControl,
                 method = "rf",
                 tuneGrid = expand.grid(mtry = num_var[1]),
                 metric = "ROC")

  yhat_rf = predict(mod_rf, spam_test[,-58])

  err_mat[r, 2] = mean(yhat_rf!=spam_test[,58])

  cm_rf <- confusionMatrix(yhat_rf, spam_test[,58], positive = "No")
  cm_rf_list[[r]] = cm_rf

  sens_mat[r, 2] = cm_rf$byClass["Sensitivity"]
```

```
  spec_mat[r, 2] = cm_rf$byClass["Specificity"]

  # just a nice statement to tell you when each loop is done
  cat("Finished Rep",r, "\n")
}
```

Find confusion matrices for the best and worst relative run of the random forests

```
# Divide error rates into two parts, corresponding to base and optimized
# models
err_reg <- slice(err_mat_melt_final, 1:50)
err_opt <- slice(err_mat_melt_final, 51:100)

# Determine "best case" repetition for which optimized model most
# outperforms base model
which.max(err_reg$Error - err_opt$Error)
```

```
## [1] 43
```

```
# Determine "worst case" repetition for which optimized model most
# underperforms base model
which.min(err_reg$Error - err_opt$Error)
```

```
## [1] 12
```

```
# Print base model confusion matrix for "best case"
cm_reg_list[[43]]
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1065   46
##        Yes   50  679
##
##                Accuracy : 0.9478
##                  95% CI : (0.9367, 0.9575)
##     No Information Rate : 0.606
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8908
##  Mcnemar's Test P-Value : 0.7595
##
##             Sensitivity : 0.9552
##             Specificity : 0.9366
##          Pos Pred Value : 0.9586
##          Neg Pred Value : 0.9314
##              Prevalence : 0.6060
##          Detection Rate : 0.5788
##    Detection Prevalence : 0.6038
##       Balanced Accuracy : 0.9459
##
##        'Positive' Class : No
##
```

```
# Print optimized model confusion matrix for "best case"
cm_rf_list[[43]]
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1078   46
##        Yes   37  679
##
##                Accuracy : 0.9549
##                  95% CI : (0.9444, 0.9639)
##     No Information Rate : 0.606
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9053
##  Mcnemar's Test P-Value : 0.3799
##
##             Sensitivity : 0.9668
##             Specificity : 0.9366
##          Pos Pred Value : 0.9591
##          Neg Pred Value : 0.9483
##              Prevalence : 0.6060
##          Detection Rate : 0.5859
##    Detection Prevalence : 0.6109
##       Balanced Accuracy : 0.9517
##
##        'Positive' Class : No
##
```

```
# Print base model confusion matrix for "worst case"
cm_reg_list[[12]]
```

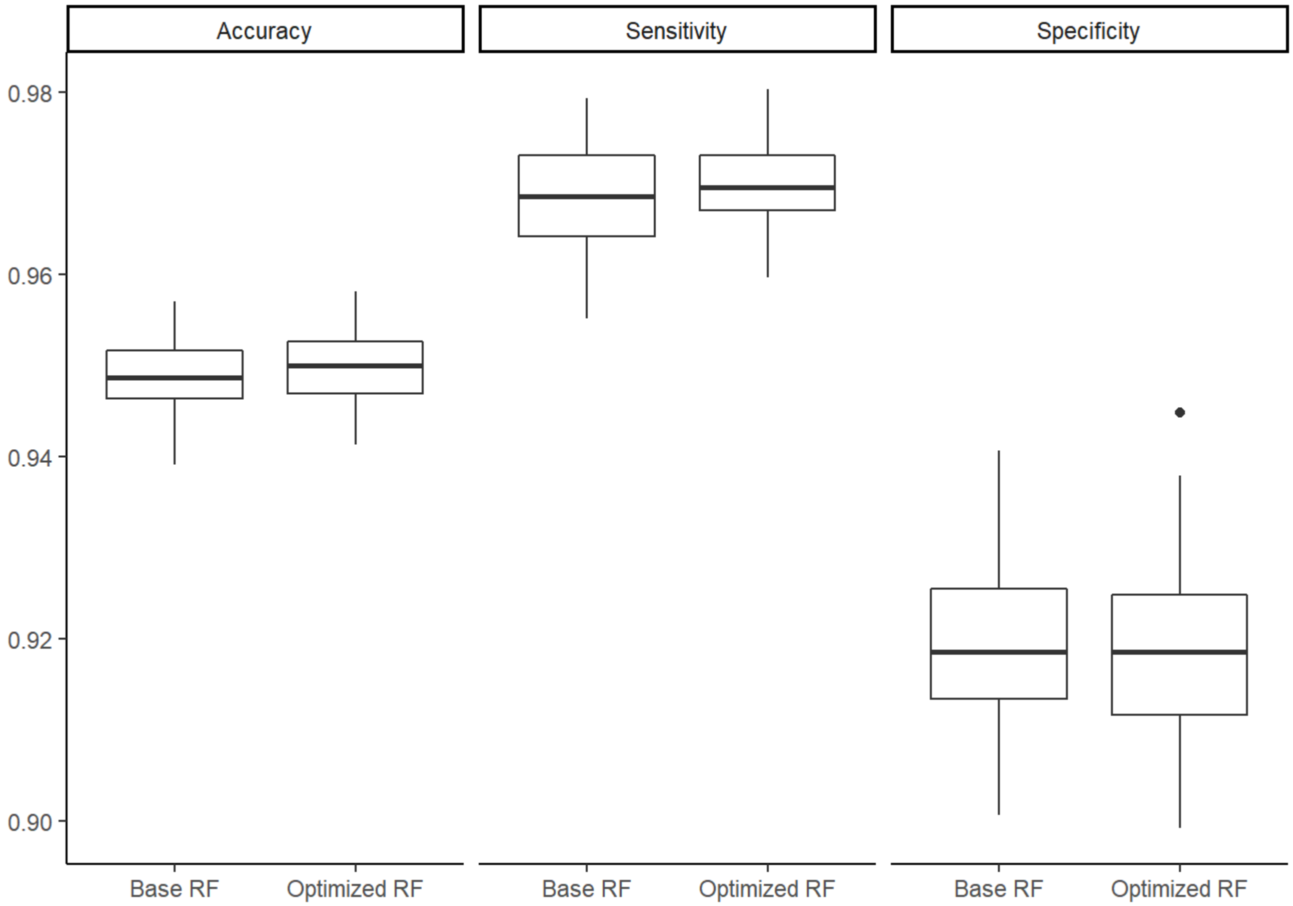
```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   No  Yes
##        No  1078   60
##        Yes   37  665
##                                           
##                Accuracy : 0.9473          
##                  95% CI : (0.9361, 0.957)
##     No Information Rate : 0.606           
##     P-Value [Acc > NIR] : <2e-16          
##                                           
##                   Kappa : 0.889           
##  Mcnemar's Test P-Value : 0.0255          
##                                           
##             Sensitivity : 0.9668          
##             Specificity : 0.9172          
##          Pos Pred Value : 0.9473          
##          Neg Pred Value : 0.9473          
##              Prevalence : 0.6060          
##          Detection Rate : 0.5859          
##    Detection Prevalence : 0.6185          
##       Balanced Accuracy : 0.9420          
##                                           
##        'Positive' Class : No              
## 
```

```
# Print optimized model confusion matrix for "worst case"
cm_rf_list[[12]]
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  1077   70
##        Yes   38  655
##
##                Accuracy : 0.9413
##                  95% CI : (0.9296, 0.9516)
##     No Information Rate : 0.606
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8761
##  Mcnemar's Test P-Value : 0.002855
##
##             Sensitivity : 0.9659
##             Specificity : 0.9034
##          Pos Pred Value : 0.9390
##          Neg Pred Value : 0.9452
##              Prevalence : 0.6060
##          Detection Rate : 0.5853
##    Detection Prevalence : 0.6234
##       Balanced Accuracy : 0.9347
##
##        'Positive' Class : No
##
```

Graph the final optimized vs regular random forest boxplot

```r
#turn error into accuracy
err_mat_melt_final$Accuracy <- 1 - err_mat_melt_final$Error

#combine the metric outputs
err_mat_final <- bind_rows(err_mat_melt_final,
                           sens_mat_melt_final,
                           spec_mat_melt_final)

#recode the method
err_mat_final$Method <- ifelse(err_mat_final$Method == "V1",
                               "Base RF",
                               "Optimized RF")

#add an indicator column
err_mat_final$Metric <- c(rep("Accuracy", 100),
                          rep("Sensitivity", 100),
                          rep("Specificity", 100))

#get metric values into a single column
err_mat_final$Value <- coalesce(err_mat_final$Accuracy,
                                err_mat_final$Sensitivity,
                                err_mat_final$Specificity)

##check it
#head(err_mat_final)
#tail(err_mat_final)


#plot
ggplot(
  data = err_mat_final,
  aes(x = Method, y = Value)
) + geom_boxplot() +
  theme_classic() +
  facet_wrap(~Metric) +
  ylab("") + xlab("")
```


Accuracy
Sensitivity
Specificity
0.98
0.96
0.94
0.92
0.90
Base RF
Optimized RF
Base RF
Optimized RF
Base RF
Optimized RF