Random For(r)est

Sebastian Heilmann

19 8 2020

In this script, we will use linguistic features to predict the speaker (Forrest Gump vs anyone else) of spoken lines in the movie “Forrest Gump” by Robert Zemeckis.

1 Explore data

Starting with exploratory data analysis is always important before modeling. Let’s start by exploring the data set and some sentences spoken by Forrest or someone else.

```
as.tbl(head(forrest))
```

```
## # A tibble: 6 x 32
##   run   onset duration offset time_to_next overlapping person FG    kind
##   <chr> <dbl>    <dbl>  <dbl> <chr>        <chr>       <chr>  <chr> <chr>
## 1 run-1  1207     1.4    2607 1.96         False       ERZAE… 1     spee…
## 2 run-1  3167     3.84   7007 4.16         False       ERZAE… 1     spee…
## 3 run-1  7327     1.6    8927 2.08         False       ERZAE… 1     spee…
## 4 run-1  9407     0.68  10087 6.6          False       ERZAE… 1     spee…
## 5 run-1 16007     3.48  19487 4.76         False       ERZAE… 1     spee…
## 6 run-1 20767   145.   165567 2.08         False       OST    1     song
## # … with 23 more variables: prosody <chr>, sentence_grammar <chr>, text <chr>,
## #   comp_score <dbl>, word_count <int>, lexfreq_norm_log_sum <dbl>,
## #   lexfreq_norm_log_mean <dbl>, lexmin <dbl>, lexmax <dbl>, sent_sum <dbl>,
## #   isforrest <chr>, satzid <fct>, open <int>, closed <int>, undefined <int>,
## #   document <chr>, CTTR <dbl>, n_lowers <dbl>, n_caps <dbl>, n_uq_words <dbl>,
## #   n_uq_chars <dbl>, n_charsperword <dbl>, depth <dbl>
```

```
set.seed(51)
forrest %>%
  filter(!is.na(isforrest)) %>%
  group_by(isforrest) %>%
  sample_n(4) %>%
  select(isforrest, text)
```

```
## # A tibble: 8 x 2
## # Groups:   isforrest [2]
##   isforrest text
##   <chr>     <chr>
## 1 FORREST   Wir waren wie eine Familie
## 2 FORREST   Und das Abendessen
## 3 FORREST   Wir waren wie Pech und Schwefel
## 4 FORREST   Sie wollte dass ich die beste Schulbildung bekomme
## 5 OTHER     Es ist hell
## 6 OTHER     Everybodys Talkin
## 7 OTHER     Denk immer an das was ich dir gesagt habe
## 8 OTHER     Sie trägt noch ihr türkisfarbenes Kellnerinnen-Kleid
```

We have a lot of features! But let’s try to take the sentence length into account and calculate the speak rate (words per seconds).

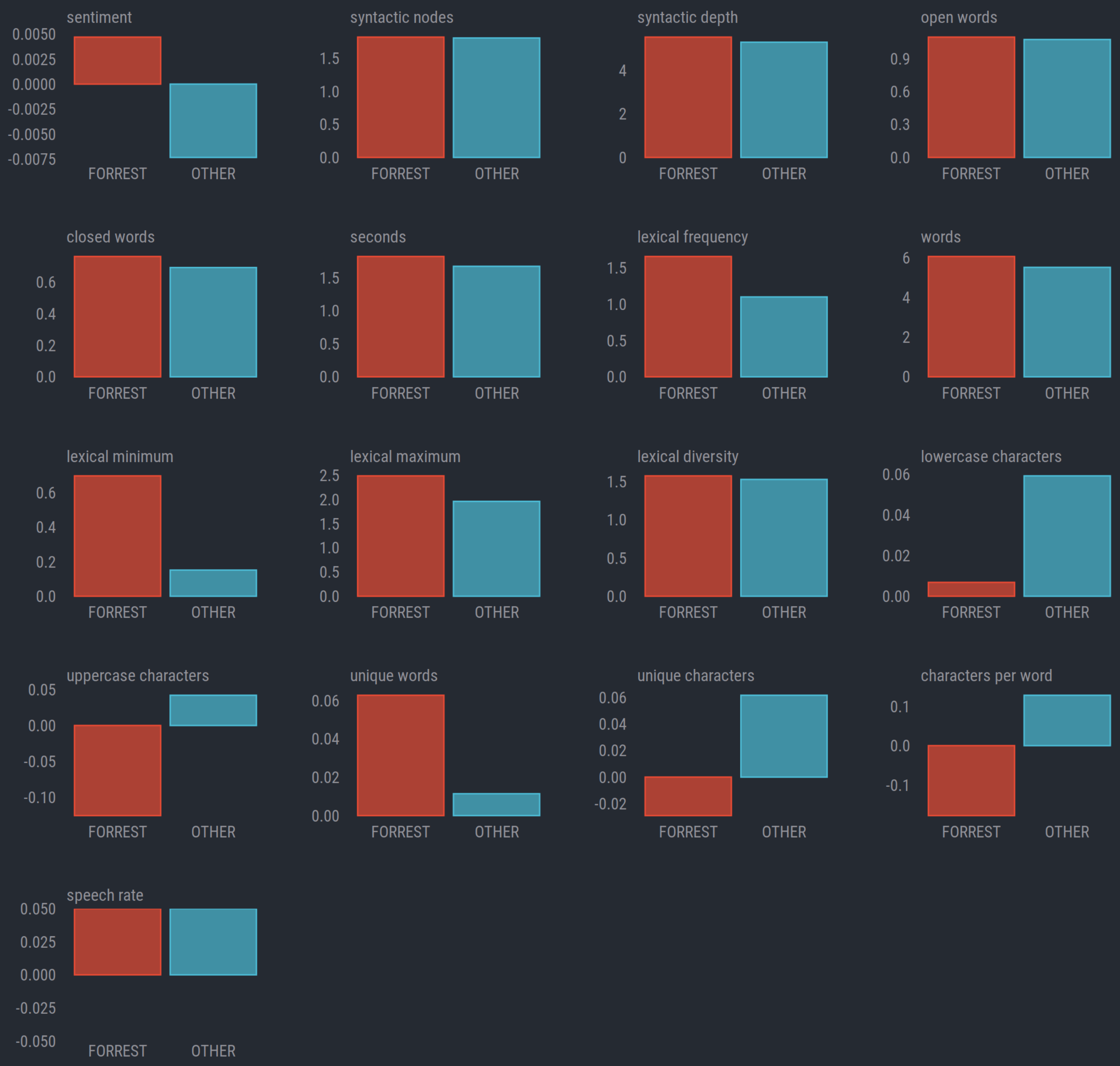
```
forrest <- forrest %>%
  mutate(sentiment = sent_sum / word_count,
         nodes = comp_score / word_count,
         open_words = open / word_count,
         closed_words = closed / word_count,
         speech_rate = word_count / duration)
```

Now its time to make a nice plot to explore the relationships in this data. Are there any text features that distinguish Forrest from other speakers?

```r
forrest %>% # we will discard "sentences" which are shown as OST in the data set
  filter(duration < max(forrest[forrest$duration > 10,c("duration","isforrest","text")] %>%
                        filter(isforrest == "FORREST") %>%
                        select(duration))) %>%
  group_by(isforrest) %>%
  filter(!is.na(isforrest)) %>%
  summarise(sentiment = mean(sentiment, na.rm = T),
            `syntactic nodes` = mean(nodes, na.rm = T),
            `syntactic depth` = mean(depth, na.rm = T),
            `open words` = mean(open_words),
            `closed words` = mean(closed_words),
            seconds = mean(duration, na.rm = T),
            `lexical frequency` = mean(lexfreq_norm_log_mean, na.rm = T),
            words = mean(word_count),
            `lexical minimum` = mean(lexmin, na.rm = T),
            `lexical maximum` = mean(lexmax, na.rm = T),
            `lexical diversity` = mean(CTTR, na.rm = T),
            `lowercase characters` = mean(n_lowers, na.rm = T),
            `uppercase characters` = mean(n_caps, na.rm = T),
            `unique words` = mean(n_uq_words, na.rm = T),
            `unique characters` = mean(n_uq_chars, na.rm = T),
            `characters per word` = mean(n_charsperword, na.rm = T),
            `speech rate` = mean(speech_rate, na.rm = T)) %>%
  reshape2::melt() %>%
  ggplot(aes(x = isforrest, y = value, color = isforrest, fill = isforrest)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~variable, scales = "free", ncol = 4) +
  labs(title = "Overview of predictors",
       subtitle = "Mean text predictors per sentence") +
  ylab("") +
  labs(x = NULL) +
  scale_fill_npg(alpha = 0.7) +
  scale_color_npg() +
  theme_ft_rc() +
  theme(panel.border = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Overview of predictors

Mean text predictors per sentence



We can see differences in lexical features and character-features especially. We will now create a seperate data set with the variables shown in the plot above.

```
# preprocess the data set
forrest_clean <- forrest %>%
                 # any sentence above ~13 is not a real sentence, therefore we will discard those
                 filter(duration < max(forrest[forrest$duration > 10, c("duration","isforrest","text")] %>%
                 # remove two sentences where the speaker is not defined
                 filter(isforrest == "FORREST") %>%
                 select(duration)))
```

```
# select relevant variables
forrest_clean <- forrest_clean %>%
  select(c(CTTR:speech_rate, duration, word_count, lexfreq_norm_log_mean:lexmax,isforrest)) %>%
  filter(!is.infinite(speech_rate)) %>%
  na.omit()
```

2 Build a random forest model

2.1 Split data into training and test data set

Our first step is to split our data into a training and testing data set. We will then create cross-validation resamples of the training data to evaluate our random forest models.

```
set.seed(123)
forrest_split <- initial_split(forrest_clean, strata = isforrest)
forrest_train <- training(forrest_split)
forrest_test <- testing(forrest_split)

set.seed(123)
forrest_folds <- vfold_cv(forrest_train, strata = isforrest)
```

2.2 Define a tidymodel recipe and workflow.

We now want to tune some hyperparameters. Because we use a random forest model, we don't need to do much preprocessing. That means, that we don't need to worry about centering or scaling our data.

```
forrest_recipe <-
  recipe(isforrest ~ ., data = forrest_clean) %>%
  step_naomit(all_predictors()) %>%
  step_zv(all_predictors())

rf_model <-
  # specify that the model is a random forest
  rand_forest() %>%
  # specify parameters that we want to tune
  set_args(mtry = tune(), trees = tune(), min_n = tune()) %>%
  # select the engine that underlies the model
  set_engine("ranger", importance = "impurity") %>%
  # choose binary classification mode
  set_mode("classification")
rf_model
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
##
## Engine-Specific Arguments:
##   importance = impurity
##
## Computational engine: ranger
```

```
rf_workflow <- workflow() %>%
  # add the recipe
  add_recipe(forrest_recipe) %>%
  # add the model
  add_model(rf_model)

rf_workflow
```

```
## ══ Workflow ════════════════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: rand_forest()
##
## ── Preprocessor ────────────────────────────────────────────────────────────────────────
## 2 Recipe Steps
##
## ● step_naomit()
## ● step_zv()
##
## ── Model ───────────────────────────────────────────────────────────────────────────────
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
##
## Engine-Specific Arguments:
##   importance = impurity
##
## Computational engine: ranger
```

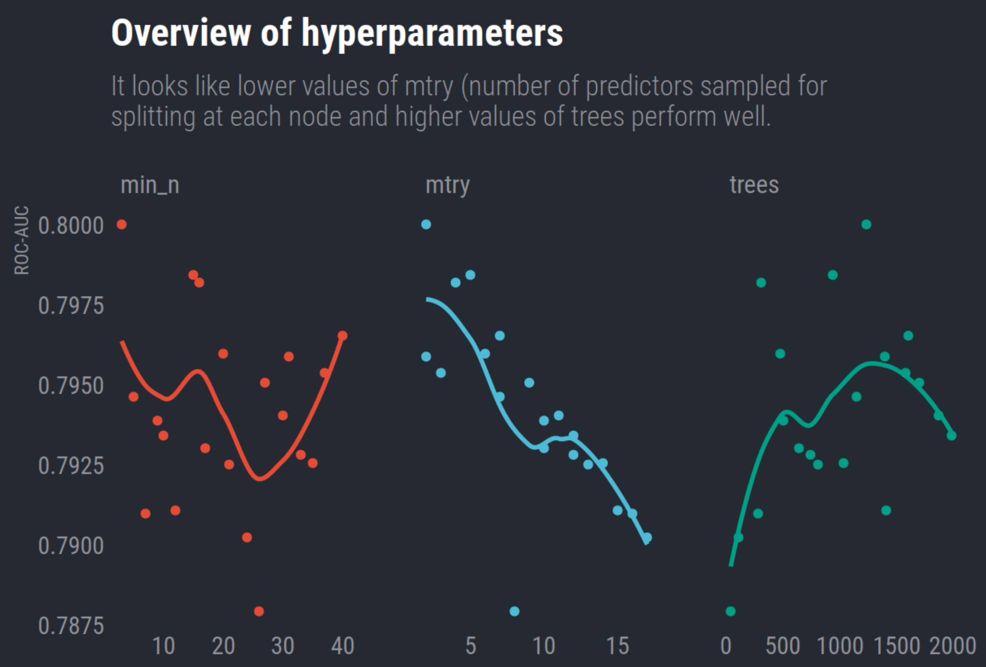
2.3 Tune hyperparameters

We will use tune_grid() with our tuneable workflow (and our grid of parameters (mtry, trees, min_n) and our resamples to try. Most importantly, but optional, we will call doParallel::registerDoParallel() to parallelize R code on our machine.

```
doParallel::registerDoParallel()

set.seed(234)
tune_res <- tune_grid(
  rf_workflow,
  resamples = forrest_folds,
  grid = 20
)
```

Let's now try a visualization to understand our results.



What are the best performing sets of parameters?

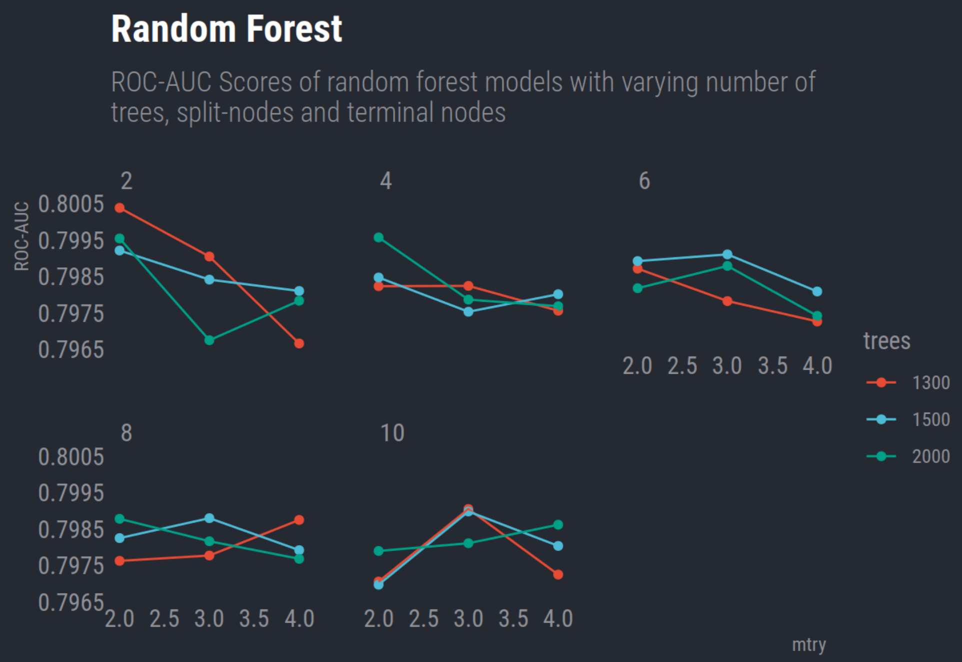
```
show_best(tune_res, "roc_auc")
```

```
## # A tibble: 5 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2  1236     3 roc_auc binary     0.800    10 0.00649 Model12
## 2     5   940    15 roc_auc binary     0.798    10 0.00644 Model11
## 3     4   310    16 roc_auc binary     0.798    10 0.00620 Model03
## 4     7  1605    40 roc_auc binary     0.797    10 0.00623 Model01
## 5     6   478    20 roc_auc binary     0.796    10 0.00648 Model05
```

We will tune our hyperparameters once more, but in a more fine grained way.

```
set.seed(123)
# set tuning parameters to try
rf_grid <- expand.grid(mtry = c(2,3,4), trees = c(1300, 1500, 2000), min_n = c(4,6,8,10,20,30,32,34,36))
# apply model and extract results
rf_tune_results <- rf_workflow %>%
  tune_grid(resamples = forrest_folds,
            grid = rf_grid,
            # choose some metrics
            metrics = metric_set(accuracy, roc_auc, sens, spec)
  )
```

Let's have a look at all combinations of parameters.



3 Finalize the workflow and evaluate the test set

```
param_final <- rf_tune_results %>% select_best(metric = "roc_auc")
rf_workflow <- rf_workflow %>% finalize_workflow(param_final)
```

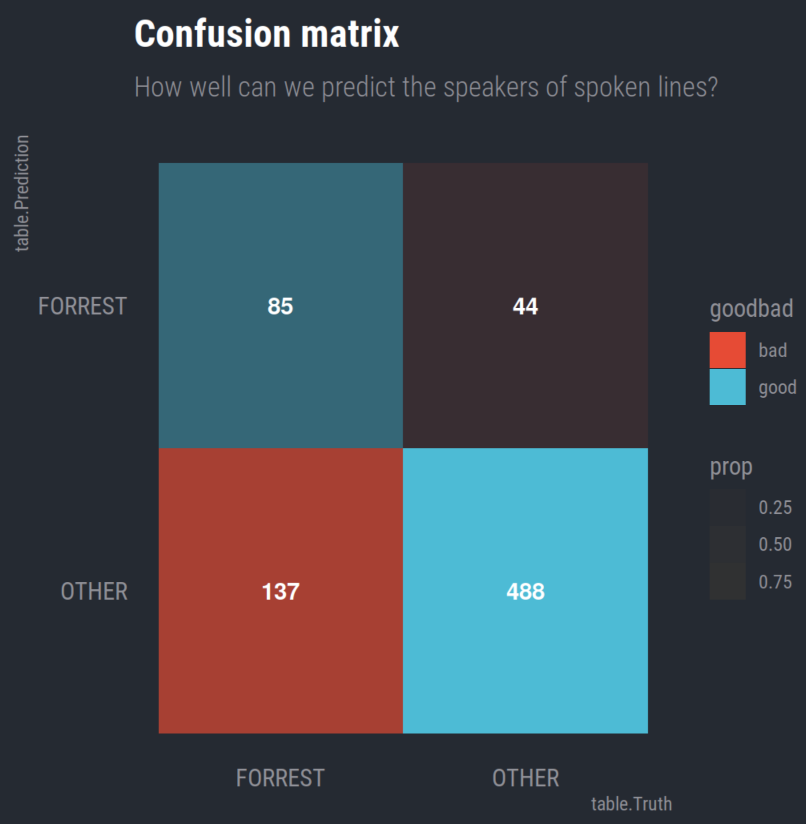
```
# fit on the training set and evaluate on test set
rf_fit <- rf_workflow %>% last_fit(forrest_split)
# check performance on test set
test_performance <- rf_fit %>% collect_metrics()
# generate predictions from the test set
test_predictions <- rf_fit %>% collect_predictions()
```

One way to visualize the perfomance of a classification model is to use a confusion matrix. The confusion matrix shows the predictions the model got correct (top left and bottom right corners) and the predictions missed by the model (lower left and upper right corners). We can also create a ROC curve for the testing set.

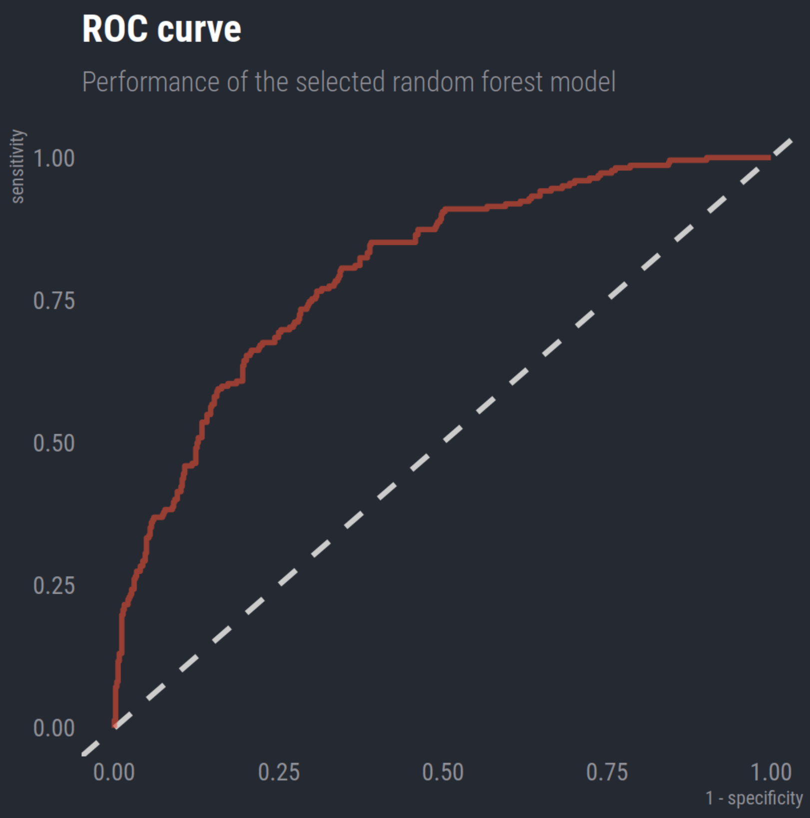
```
# create confusion matrix
confusion_matrix <- test_predictions %>% conf_mat(truth = isforrest, estimate = .pred_class)
confusion_matrix <- as.data.frame(confusion_matrix["table"])

plotTable <- confusion_matrix %>%
  mutate(goodbad = ifelse(confusion_matrix$table.Prediction == confusion_matrix$table.Truth, "good", "bad")) %>%
  group_by(table.Truth) %>%
  mutate(prop = table.Freq/sum(table.Freq))

ggplot(data = plotTable, mapping = aes(x = table.Truth, y = table.Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = table.Freq), vjust = .5, fontface  = "bold", alpha = 1, color = "white") +
  theme_minimal() +
  ylim(rev(levels(confusion_matrix$table.Prediction))) +
  theme_ft_rc() +
  labs(title = "Confusion matrix",
       subtitle = "How well can we predict the speakers of spoken lines?") +
  theme(panel.border = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  scale_fill_npg(alpha = 1) +
  scale_color_npg()
```




```
test_predictions %>%
  group_by(id) %>%
  roc_curve(isforrest, .pred_FORREST) %>%
  ggplot(aes(1 - specificity, sensitivity, color = id)) +
  geom_abline(lty = 2, color = "gray80", size = 1.2) +
  geom_path(show.legend = FALSE, alpha = 0.6, size = 1.2) +
    labs(title = "ROC curve",
         subtitle = "Performance of the selected random forest model") +
    theme_minimal() +
    theme_ft_rc() +
    theme(panel.border = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank()) +
    scale_fill_npg() +
    scale_color_npg()
```

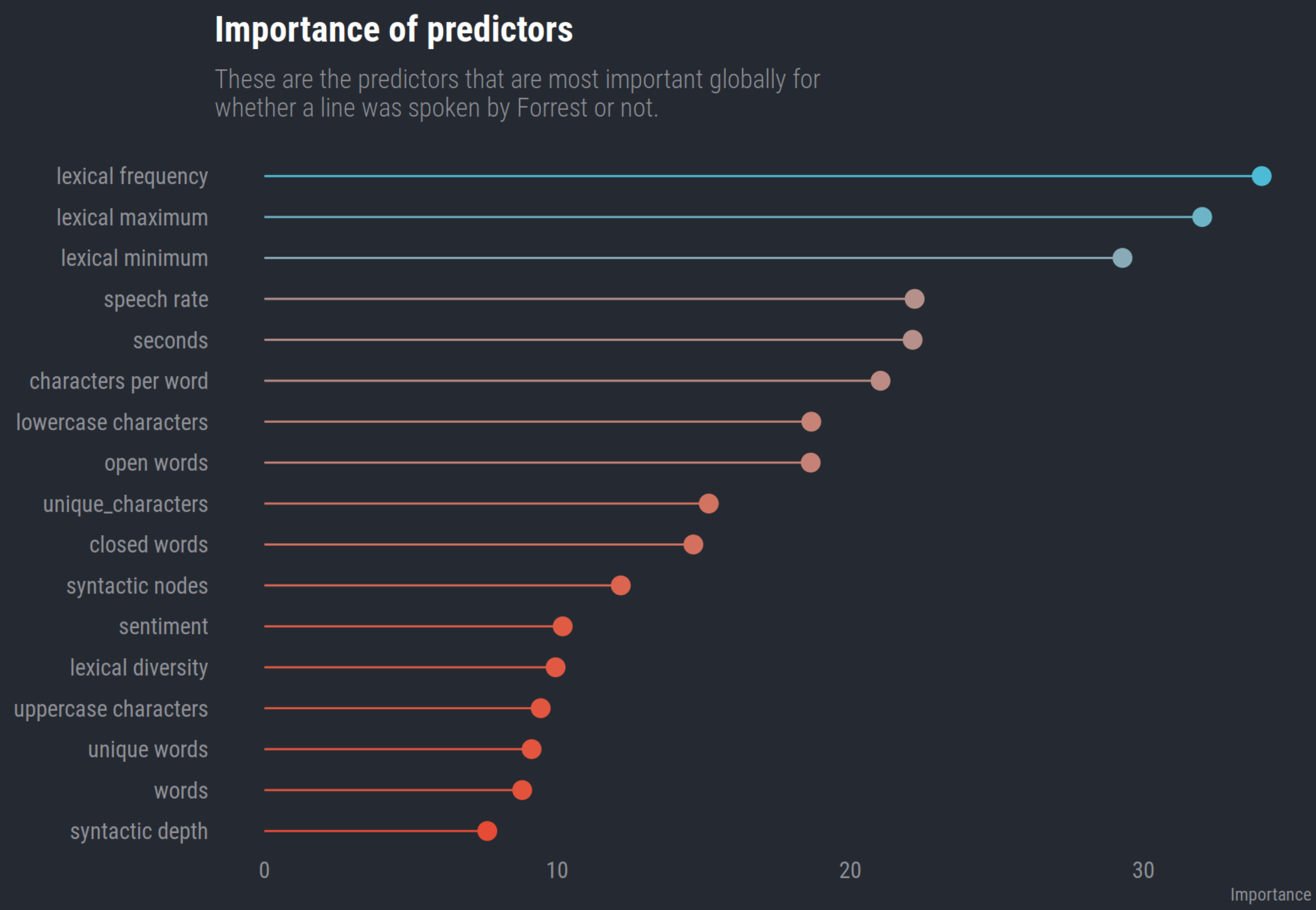


Our results indicate that we did not overfit during the tuning process. The performance of our model is moderate. Which features were the best to distinguish speakers of spoken lines?

```
library(vip)
imp_df <- rf_workflow %>%
  fit(data = forrest_test) %>%
  pull_workflow_fit() %>%
  vi()

# rename variables for plotting
imp_df$Variable <- c("lexical frequency",
                     "lexical maximum",
                     "lexical minimum",
                     "speech rate",
                     "seconds",
                     "characters per word",
                     "lowercase characters",
                     "open words",
                     "unique_characters",
                     "closed words",
                     "syntactic nodes",
                     "sentiment",
                     "lexical diversity",
                     "uppercase characters",
                     "unique words",
                     "words",
                     "syntactic depth")

imp_df$Variable <- as.factor(imp_df$Variable)
```



4 Conclusion

Overall our model did a fairly good job. As mentioned in the beginning of our exploratory analyses, lexical features seems to be most predictive. Our model may be further improved by taking the content of spoken words or topics of sentences into account.