Slay: Predicting song artist based on lyrics

Suggested answers

APPLICATION EXERCISE ANSWERS

MODIFIED
August 13, 2024

```
library(tidyverse)
library(tidymodels)
library(stringr)
library(textrecipes)
library(textdata)
library(discrim)
library(themis)
library(vip)

# set seed for randomization
set.seed(123)

theme_set(theme_minimal(base_size = 13))
```

Import data

```
lyrics <- read_csv(file = "data/beyonce-swift-lyrics.csv") |>
  mutate(artist = factor(artist))
lyrics
```

```
# A tibble: 355 × 19
   album_name track_number track_name artist lyrics danceability energy loudness
   <chr>             <dbl> <chr>      <fct>  <chr>         <dbl>  <dbl>    <dbl>
 1 COWBOY CA…            1 AMERIICAN… Beyon… "Noth…        0.374  0.515    -6.48
 2 COWBOY CA…            3 16 CARRIA… Beyon… "Sixt…        0.525  0.456    -7.04
 3 COWBOY CA…            5 MY ROSE    Beyon… "How …        0.384  0.177   -11.8
 4 COWBOY CA…            7 TEXAS HOL… Beyon… "This…        0.727  0.711    -6.55
 5 COWBOY CA…            8 BODYGUARD  Beyon… "One,…        0.726  0.779    -5.43
 6 COWBOY CA…           10 JOLENE     Beyon… "(Jol…        0.567  0.812    -5.43
 7 COWBOY CA…           11 DAUGHTER   Beyon… "Your…        0.374  0.448   -10.0
 8 COWBOY CA…           13 ALLIIGATO… Beyon… "High…        0.618  0.651    -9.66
 9 COWBOY CA…           18 FLAMENCO   Beyon… "My m…        0.497  0.351    -9.25
10 COWBOY CA…           20 YA YA      Beyon… "Hell…        0.617  0.904    -5.37
# ℹ 345 more rows
# ℹ 11 more variables: speechiness <dbl>, acousticness <dbl>,
#   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
```

```
#   time_signature <dbl>, duration_ms <dbl>, explicit <lgl>, key_name <chr>,
#   mode_name <chr>
```

Split the data into analysis/assessment/test sets

Demonstration:

- Split the data into training/test sets with 75% allocated for training
- Split the training set into 10 cross-validation folds

```
# split into training/testing
set.seed(123)
lyrics_split <- initial_split(data = lyrics, strata = artist, prop = 0.75)

lyrics_train <- training(lyrics_split)
lyrics_test <- testing(lyrics_split)

# create cross-validation folds
lyrics_folds <- vfold_cv(data = lyrics_train, strata = artist)
```

Estimate the null model for a baseline comparison

Demonstration: Estimate a null model to determine an appropriate baseline for evaluating a model's performance.

```
null_spec <- null_model() |>
  set_engine("parsnip") |>
  set_mode("classification")

null_spec |>
  fit_resamples(
    # pick something as the predictor - doesn't really matter
    artist ~ danceability,
    resamples = lyrics_folds
  ) |>
  collect_metrics()
```

```
# A tibble: 3 × 6
  .metric     .estimator  mean     n  std_err .config
  <chr>       <chr>      <dbl> <int>    <dbl> <chr>
1 accuracy    binary     0.668    10 0.00236  Preprocessor1_Model1
2 brier_class binary     0.222    10 0.000792 Preprocessor1_Model1
3 roc_auc     binary     0.5      10 0        Preprocessor1_Model1
```

Fit a random forest model

Define the feature engineering recipe

Demonstration:

- Define a feature engineering recipe to predict the song's artist as a function of the lyrics + audio features
- Exclude the ID variables from the recipe
- Tokenize the song lyrics
- Remove stop words
- Only keep the `500` most frequently appearing tokens
- Calculate tf-idf scores for the remaining tokens
    - This will generate one column for every token. Each column will have the standardized name `tfidf_lyrics_*` where `*` is the specific token. Instead we would prefer the column names simply be `*`. You can remove the `tfidf_lyrics_` prefix using

    ```
    # Simplify these names
    step_rename_at(starts_with("tfidf_lyrics_"),
      fn = \(x) str_replace_all(
        string = x,
        pattern = "tfidf_lyrics_",
        replacement = ""
      )
    )
    ```

    - This does cause a conflict between the `energy` audio feature and the token `energy`. Before removing the `"tfidf_lyrics_"` prefix, we will add a prefix to the audio features to avoid this conflict.

    ```
    # Simplify these names
    step_rename_at(
      all_predictors(), -starts_with("tfidf_lyrics_"),
      fn = \(x) str_glue("af_{x}")
    )
    ```

- Downsample the observations so there are an equal number of songs by Beyoncé and Taylor Swift in the analysis set

```
# define preprocessing recipe
rf_rec <- recipe(artist ~ ., data = lyrics_train) |>
  # exclude ID variables
  update_role(album_name, track_number, track_name, new_role = "id vars") |>
  step_tokenize(lyrics) |>
  step_stopwords(lyrics) |>
  step_tokenfilter(lyrics, max_tokens = 500) |>
  step_tfidf(lyrics) |>
  # Simplify these names
```

```
  step_rename_at(
    all_predictors(), -starts_with("tfidf_lyrics_"),
    fn = \(x) str_glue("af_{x}")
  ) |>
  step_rename_at(starts_with("tfidf_lyrics_"),
    fn = \(x) str_replace_all(
      string = x,
      pattern = "tfidf_lyrics_",
      replacement = ""
    )
  ) |>
  step_downsample(artist)
rf_rec
```

Fit the model

Demonstration:

- Define a random forest model grown with `1000` trees using the `ranger` engine.
- Define a workflow using the feature engineering recipe and random forest model specification. Fit the workflow using the cross-validation folds.
  - Use `control = control_resamples(save_pred = TRUE)` to save the assessment set predictions. We need these to assess the model's performance.
  - Use `control = control_resamples(save_workflow = TRUE)` to save the workflow object. We need this later on if we want to fit a single model using the same workflow and the entire training set.

```
# define the model specification
rf_spec <- rand_forest(trees = 1000) |>
  set_mode("classification") |>
  # calculate feature importance metrics using the ranger engine
  set_engine("ranger", importance = "permutation")

# define the workflow
rf_wf <- workflow() |>
  add_recipe(rf_rec) |>
  add_model(rf_spec)

# fit the model to each of the cross-validation folds
rf_cv <- rf_wf |>
  fit_resamples(
    resamples = lyrics_folds,
    control = control_resamples(save_pred = TRUE, save_workflow = TRUE)
  )
```

Evaluate model performance

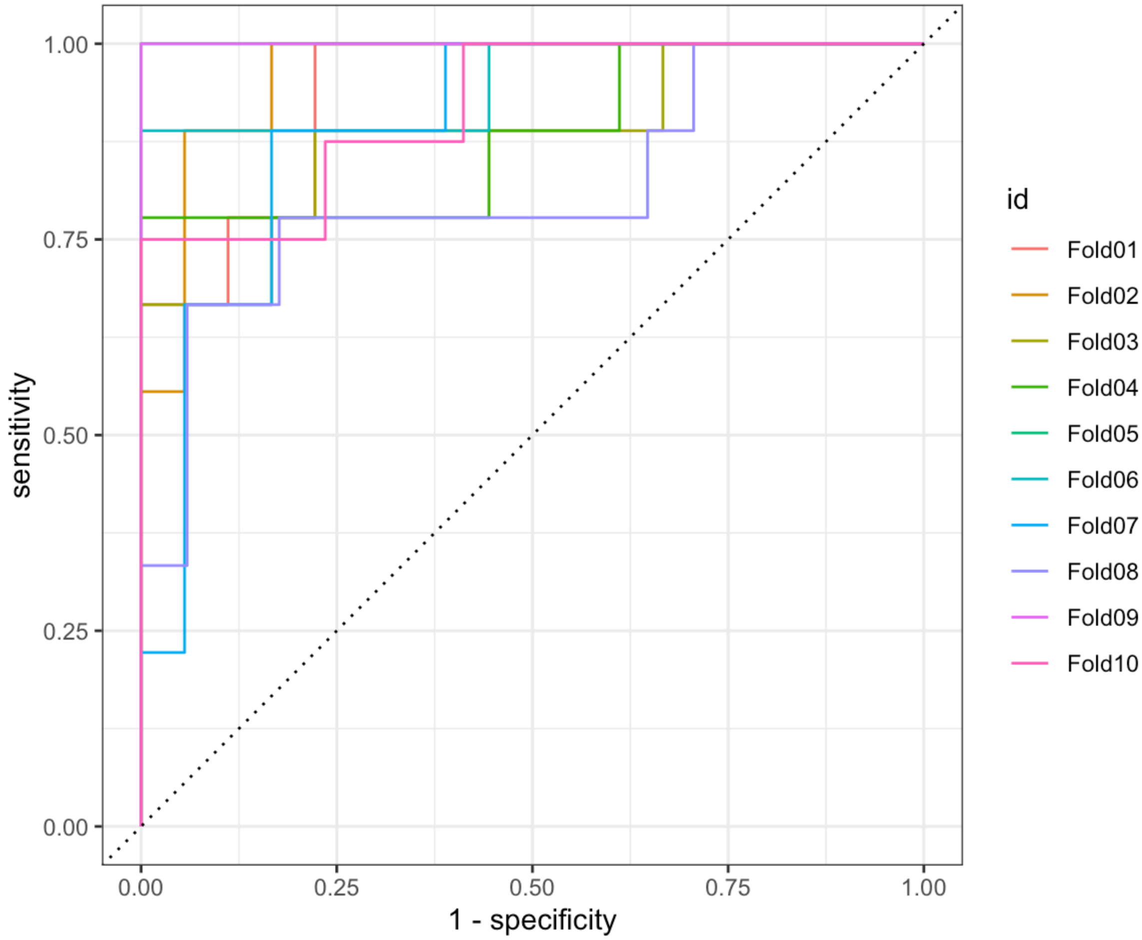
Demonstration:

- Calculate the model's accuracy and ROC AUC. How did it perform?
- Draw the ROC curve for each validation fold
- Generate the resampled confusion matrix for the model and draw it using a heatmap. How does the model perform predicting Beyoncé songs relative to Taylor Swift songs?

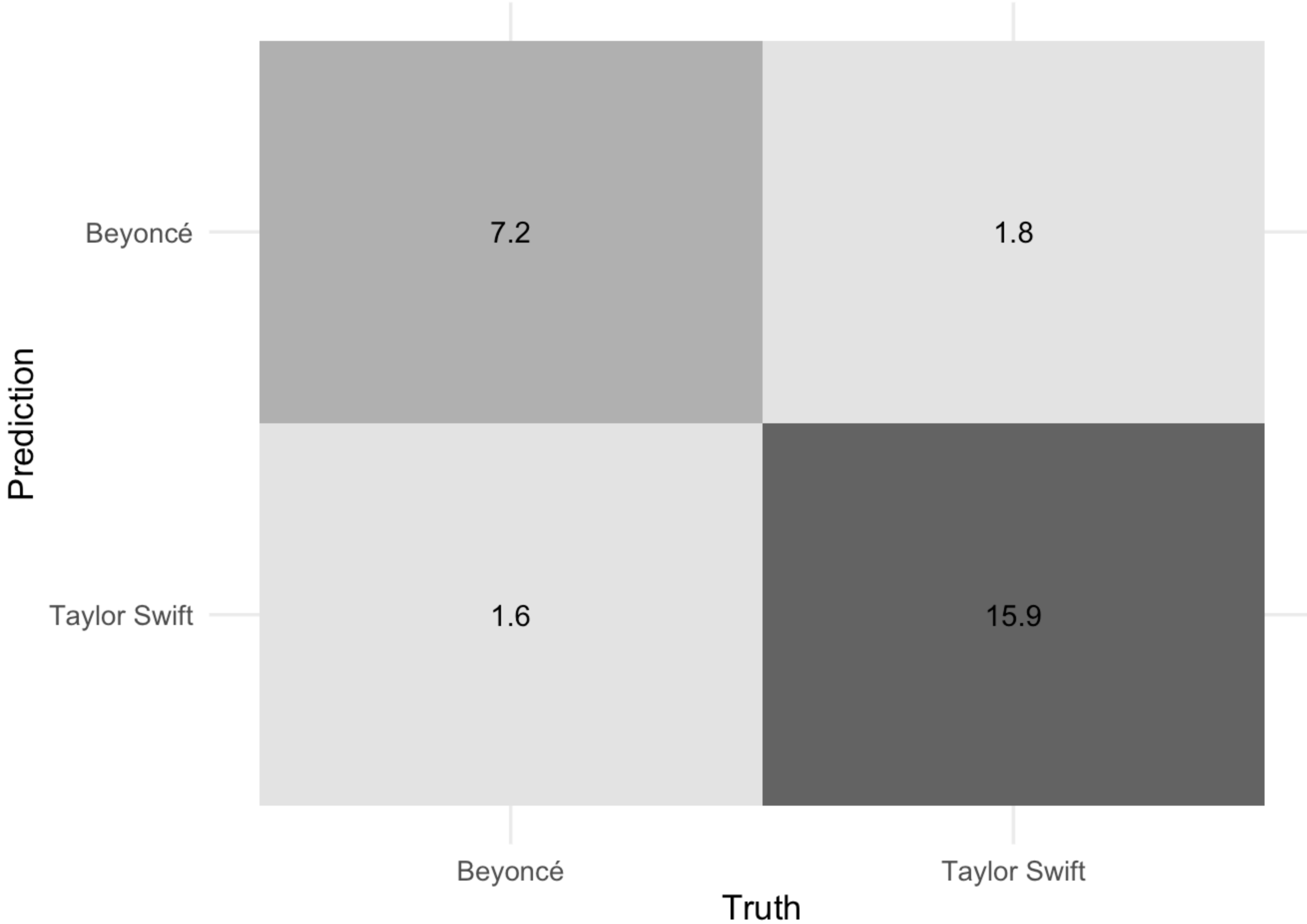
```r
# extract metrics and predictions
rf_cv_metrics <- collect_metrics(rf_cv)
rf_cv_predictions <- collect_predictions(rf_cv)

# how well did the model perform?
rf_cv_metrics
```

```
# A tibble: 3 × 6
  .metric     .estimator  mean     n std_err .config
  <chr>       <chr>      <dbl> <int>   <dbl> <chr>
1 accuracy    binary     0.872    10 0.0241  Preprocessor1_Model1
2 brier_class binary     0.155    10 0.00590 Preprocessor1_Model1
3 roc_auc     binary     0.924    10 0.0186  Preprocessor1_Model1
```

```r
# roc curve
rf_cv_predictions |>
  group_by(id) |>
  roc_curve(truth = artist, .pred_Beyoncé) |>
  autoplot()
```




```r
# confusion matrix
conf_mat_resampled(x = rf_cv, tidy = FALSE) |>
  autoplot(type = "heatmap")
```



Add response here. Overall this is a pretty strong model. The ROC AUC is quite strong. The confusion matrix shows that the model is slightly better at predicting Taylor Swift songs than Beyoncé songs, but given the overrepresentation of Taylor Swift songs in the dataset this is not unusual.

Penalized regression

Define the feature engineering recipe

Demonstration:

- Define a feature engineering recipe to predict the song's artist as a function of the lyrics + audio features
- Exclude the ID variables from the recipe
- Tokenize the song lyrics
- **Calculate all possible 1-grams, 2-grams, 3-grams, 4-grams, and 5-grams**
- Remove stop words
- Only keep the `2000` most frequently appearing tokens
- Calculate tf-idf scores for the remaining tokens
- Rename audio feature and tf-idf as before
- Apply required steps for penalized regression models
  - Convert the `explicit` variable to a factor

- Convert nominal predictors to dummy variables
- Get rid of zero-variance predictors
- Normalize all predictors to mean of 0 and variance of 1

- Downsample the observations so there are an equal number of songs by Beyoncé and Taylor Swift in the analysis set

```r
glmnet_rec <- recipe(artist ~ ., data = lyrics_train) |>
  # exclude ID variables
  update_role(album_name, track_number, track_name, new_role = "id vars") |>
  # tokenize and prep lyrics
  step_tokenize(lyrics) |>
  step_stopwords(lyrics) |>
  step_ngram(lyrics, num_tokens = 5L, min_num_tokens = 1L) |>
  step_tokenfilter(lyrics, max_tokens = 2000) |>
  step_tfidf(lyrics) |>
  # Simplify these names
  step_rename_at(
    all_predictors(), -starts_with("tfidf_lyrics_"),
    fn = \(x) str_glue("af_{x}")
  ) |>
  step_rename_at(starts_with("tfidf_lyrics_"),
    fn = \(x) str_replace_all(
      string = x,
      pattern = "tfidf_lyrics_",
      replacement = ""
    )
  ) |>
  # fix explicit variable to factor
  step_bin2factor(af_explicit) |>
  # normalize for penalized regression
  step_dummy(all_nominal_predictors()) |>
  step_zv(all_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_downsample(artist)
glmnet_rec
```

Tune the penalized regression model

Demonstration:

- Define the penalized regression model specification, including tuning placeholders for `penalty` and `mixture`
- Create the workflow object
- Define a tuning grid with every combination of:
  - `penalty = 10^seq(-6, -1, length.out = 20)`
  - `mixture = c(0, 0.2, 0.4, 0.6, 0.8, 1)`
- Tune the model using the cross-validation folds

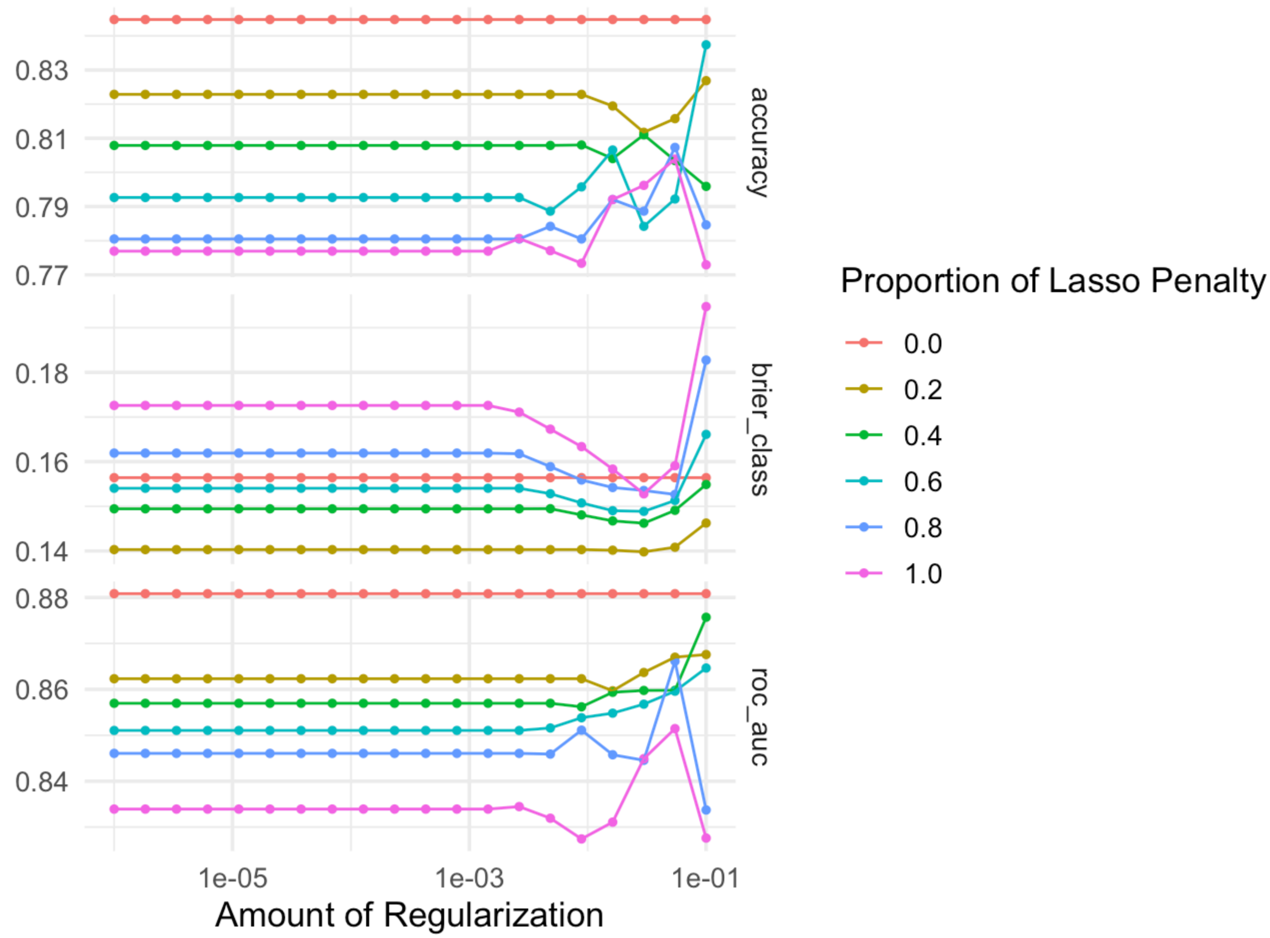
- Evaluate the tuning procedure and identify the best performing models based on ROC AUC

```r
# define the penalized regression model specification
glmnet_spec <- logistic_reg(penalty = tune(), mixture = tune()) |>
  set_mode("classification") |>
  set_engine("glmnet")

# define the new workflow
glmnet_wf <- workflow() |>
  add_recipe(glmnet_rec) |>
  add_model(glmnet_spec)

# create the tuning grid
glmnet_grid <- expand_grid(
  penalty = 10^seq(-6, -1, length.out = 20),
  mixture = c(0, 0.2, 0.4, 0.6, 0.8, 1)
)

# tune over the model hyperparameters
glmnet_tune <- tune_grid(
  object = glmnet_wf,
  resamples = lyrics_folds,
  grid = glmnet_grid,
  control = control_grid(save_pred = TRUE, save_workflow = TRUE)
)
```

```r
# evaluate results
collect_metrics(x = glmnet_tune)
```

```
# A tibble: 360 × 8
      penalty mixture .metric     .estimator  mean     n std_err .config
        <dbl>   <dbl> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
 1 0.000001         0 accuracy    binary     0.845    10 0.0186  Preprocessor1_…
 2 0.000001         0 brier_class binary     0.156    10 0.00635 Preprocessor1_…
 3 0.000001         0 roc_auc     binary     0.881    10 0.0249  Preprocessor1_…
 4 0.00000183       0 accuracy    binary     0.845    10 0.0186  Preprocessor1_…
 5 0.00000183       0 brier_class binary     0.156    10 0.00635 Preprocessor1_…
 6 0.00000183       0 roc_auc     binary     0.881    10 0.0249  Preprocessor1_…
 7 0.00000336       0 accuracy    binary     0.845    10 0.0186  Preprocessor1_…
 8 0.00000336       0 brier_class binary     0.156    10 0.00635 Preprocessor1_…
 9 0.00000336       0 roc_auc     binary     0.881    10 0.0249  Preprocessor1_…
10 0.00000616       0 accuracy    binary     0.845    10 0.0186  Preprocessor1_…
# ℹ 350 more rows
```

```r
autoplot(glmnet_tune)
```




```
# identify the five best hyperparameter combinations
show_best(x = glmnet_tune, metric = "roc_auc")
```

```
# A tibble: 5 × 8
     penalty mixture .metric .estimator  mean     n std_err .config
       <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 0.000001         0 roc_auc binary     0.881    10  0.0249 Preprocessor1_Model…
2 0.00000183       0 roc_auc binary     0.881    10  0.0249 Preprocessor1_Model…
3 0.00000336       0 roc_auc binary     0.881    10  0.0249 Preprocessor1_Model…
4 0.00000616       0 roc_auc binary     0.881    10  0.0249 Preprocessor1_Model…
5 0.0000113        0 roc_auc binary     0.881    10  0.0249 Preprocessor1_Model…
```

Add response here. The best performing hyperparameter combination has a strong ROC AUC value, though it is slightly lower compared to the random forest model.

Random forest with word embeddings

Define the feature engineering recipe

Demonstration: Rather than using individual tokens and tf-idf scores, we will use pre-trained word embeddings to represent the lyrics. We will calculate the aggregate embedding for each song by averaging the embeddings of all the words in the song, then tune a random forest model using these embeddings.

Note

Normally in order to import GloVe embeddings you would use the code below:

```
glove_embed <- embedding_glove6b(dimensions = 100)
```

This downloads the ZIP file containing the embeddings, stores it in a cache folder, and then imports the requested embeddings and dimensions as a data frame. Note that many of the embeddings are stored in ZIP files that are multiple gigabytes in size. Often it is easier to manually download the files and store them in the appropriate location outside of R. See the documentation for `embedding_glove*()` for more information.

```
# hacky way to make it work on RStudio Workbench
glove_embed <- read_delim(
    file = "/rstudio-files/glove6b/glove.6B.100d.txt",
    delim = " ",
    quote = "",
    col_names = c(
      "token",
      paste0("d", seq_len(100))
    ),
    col_types = paste0(
      c(
        "c",
        rep("d", 100)
      ),
      collapse = ""
    )
  )
```

```
# load previously-downloaded word embeddings
glove_embed <- embedding_glove6b(
  # dir = "/rstudio-files",
  dimensions = 100,
  manual_download = TRUE
)
```

```
rf_embeds_rec <- recipe(artist ~ ., data = lyrics_train) |>
  # exclude ID variables
  update_role(album_name, track_number, track_name, new_role = "id vars") |>
  step_tokenize(lyrics) |>
  # calculate aggregate embedding for each song
  step_word_embeddings(lyrics, embeddings = glove_embed, aggregation = "mean") |>
  step_downsample(artist)
rf_embeds_rec
```

Tune the model

Demonstration:

- Define the penalized regression model specification, including tuning placeholders for `mtry` and `min_n`
- Create the workflow object
- Tune the model using the cross-validation folds and an automatically generated tuning grid
- Evaluate the tuning procedure and identify the best performing models based on ROC AUC

```r
# define the model specification
rf_embeds_spec <- rand_forest(trees = 1000, mtry = tune(), min_n = tune()) |>
  set_mode("classification") |>
  # calculate feature importance metrics using the ranger engine
  set_engine("ranger", importance = "permutation")

# define the workflow
rf_embeds_wf <- workflow() |>
  add_recipe(rf_embeds_rec) |>
  add_model(rf_embeds_spec)

# fit the model to each of the cross-validation folds
rf_embeds_cv <- rf_embeds_wf |>
  tune_grid(
    resamples = lyrics_folds,
    control = control_resamples(save_pred = TRUE, save_workflow = TRUE)
  )
```

```r
# extract metrics
rf_embeds_cv_metrics <- collect_metrics(rf_embeds_cv)

# how well did the model perform?
rf_embeds_cv_metrics
```

```
# A tibble: 30 × 8
    mtry min_n .metric     .estimator  mean     n std_err .config
   <int> <int> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
 1   104    35 accuracy    binary     0.817    10 0.0278  Preprocessor1_Model01
 2   104    35 brier_class binary     0.145    10 0.00821 Preprocessor1_Model01
 3   104    35 roc_auc     binary     0.889    10 0.0231  Preprocessor1_Model01
 4    58    38 accuracy    binary     0.809    10 0.0278  Preprocessor1_Model02
 5    58    38 brier_class binary     0.146    10 0.00793 Preprocessor1_Model02
 6    58    38 roc_auc     binary     0.894    10 0.0213  Preprocessor1_Model02
 7    93    28 accuracy    binary     0.828    10 0.0309  Preprocessor1_Model03
 8    93    28 brier_class binary     0.143    10 0.00820 Preprocessor1_Model03
 9    93    28 roc_auc     binary     0.900    10 0.0204  Preprocessor1_Model03
10    45    24 accuracy    binary     0.832    10 0.0273  Preprocessor1_Model04
# ℹ 20 more rows
```

```r
show_best(rf_embeds_cv, metric = "roc_auc")
```

```
# A tibble: 5 × 8
   mtry min_n .metric .estimator  mean     n std_err .config
  <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1    78    16 roc_auc binary     0.902    10  0.0216 Preprocessor1_Model05
2    93    28 roc_auc binary     0.900    10  0.0204 Preprocessor1_Model03
3    26     8 roc_auc binary     0.900    10  0.0220 Preprocessor1_Model09
4    18    13 roc_auc binary     0.900    10  0.0204 Preprocessor1_Model10
5    90    20 roc_auc binary     0.896    10  0.0200 Preprocessor1_Model08
```

Add response here. The random forest model using word embeddings performed better than the penalized regression model, but not as well as the random forest model using tf-idf scores. That said, all the ROC AUC values are quite close to each other.

Fit the best model

Your turn:

- Select the model + hyperparameter combinations that achieve the highest ROC AUC
- Fit that model using the best hyperparameters and the full training set. How well does the model perform on the test set?

```
# select the best model's hyperparameters
best_fit <- fit_best(rf_cv)

# test set ROC AUC
bind_cols(
  lyrics_test,
  predict(best_fit, new_data = lyrics_test, type = "prob")
) |>
  roc_auc(truth = artist, .pred_Beyoncé)
```

```
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.941
```

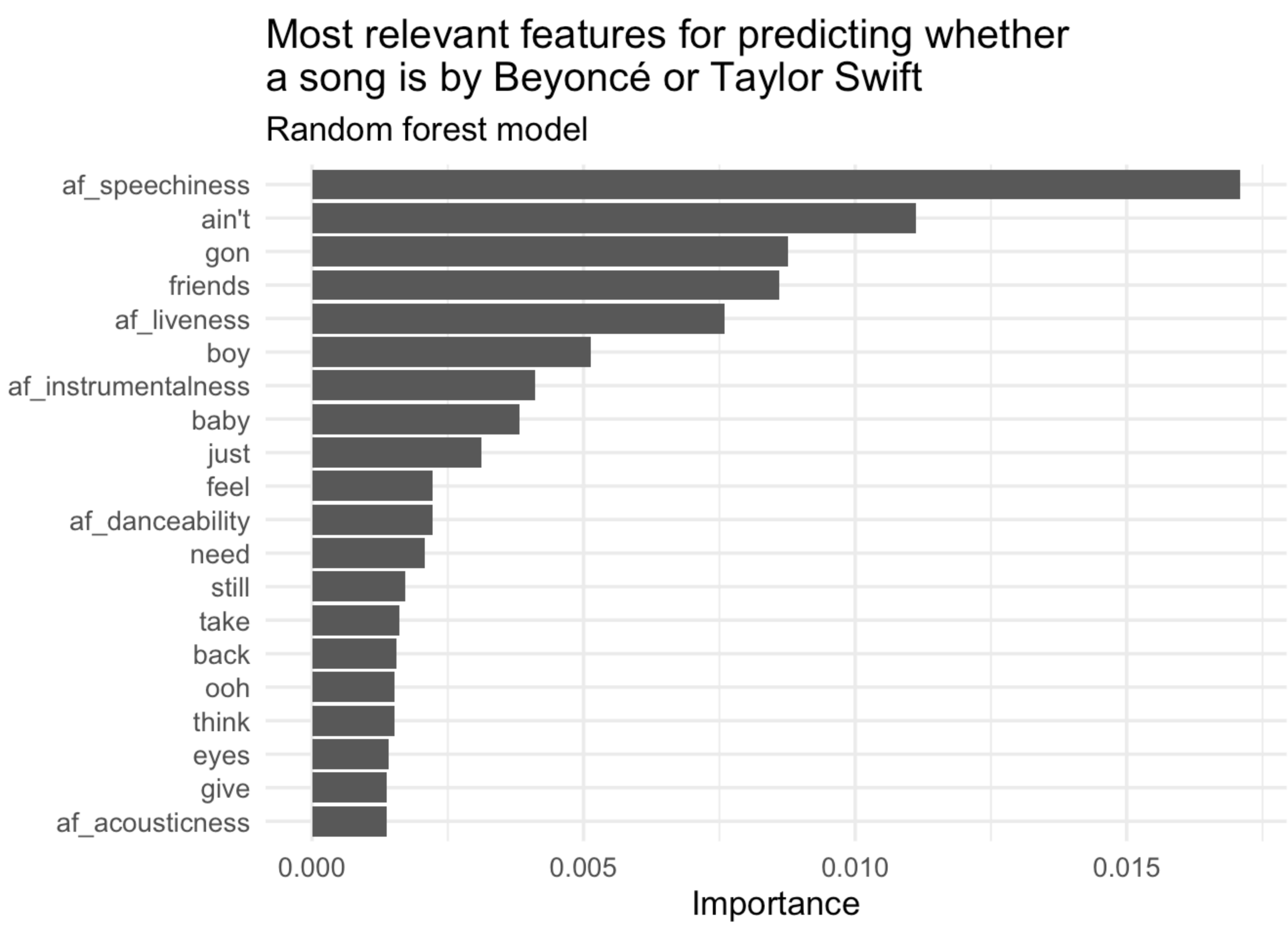
Add response here. The test set ROC AUC is slightly higher than the cross-validated metrics, indicating that the model generalizes well to new data.

Variable importance

We can examine the results of each model to evaluate which tokens were the most important in generating artist predictions. Here we use **vip** to calculate importance.

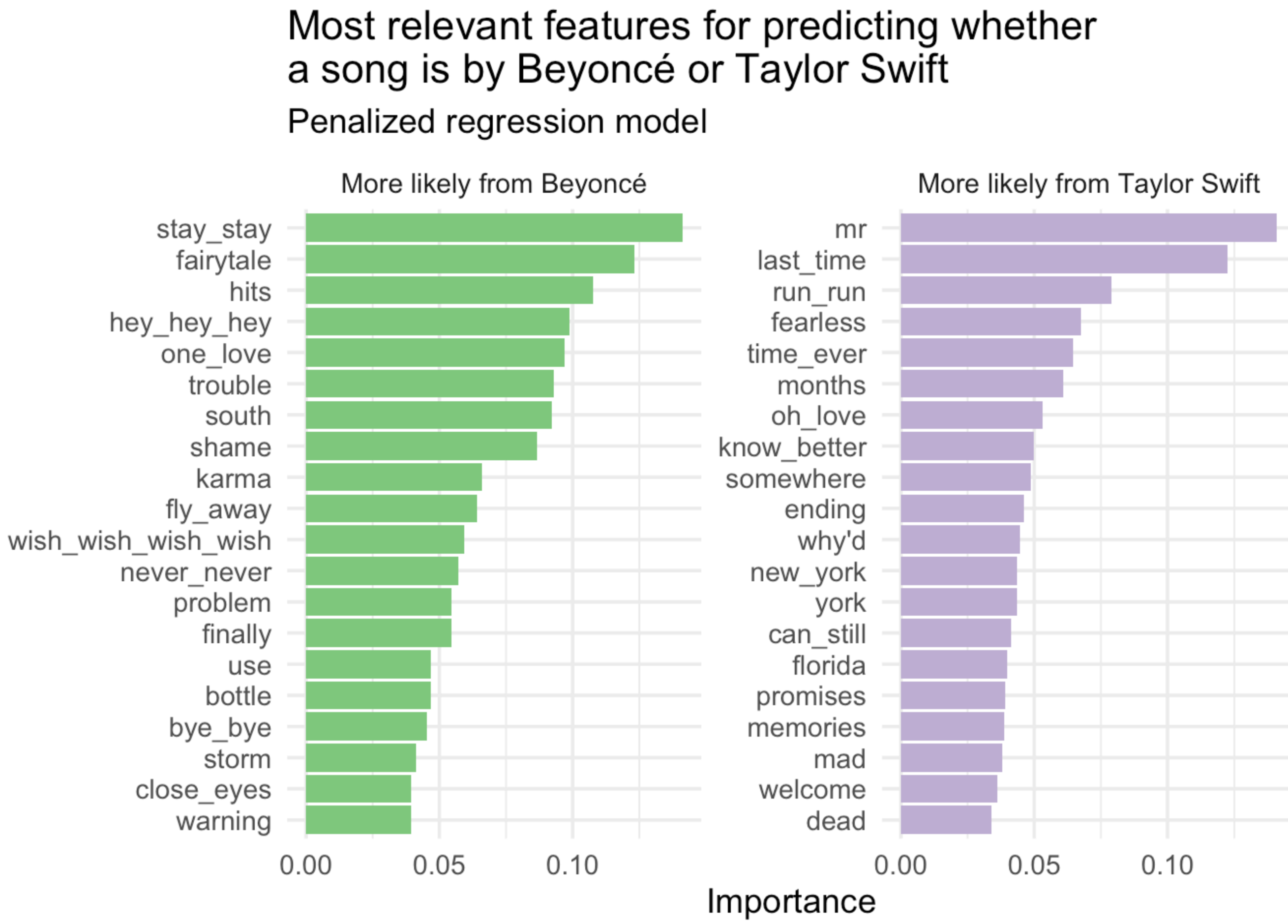
```
# extract parsnip model fit
rf_imp <- rf_cv |>
  fit_best() |>
  extract_fit_parsnip() |>
```

```r
  vi(method = "model")

# clean up the data frame for visualization
rf_imp |>
  # extract 20 most important n-grams
  slice_max(order_by = Importance, n = 20) |>
  mutate(Variable = fct_reorder(.f = Variable, .x = Importance)) |>
  ggplot(mapping = aes(
    x = Importance,
    y = Variable
  )) +
  geom_col() +
  labs(
    y = NULL,
    title = "Most relevant features for predicting whether\na song is by Beyoncé or Taylor
         Swift",
    subtitle = "Random forest model"
  )
```




```r
# extract parsnip model fit
glmnet_imp <- glmnet_tune |>
  fit_best() |>
  extract_fit_parsnip() |>
```

```
  vi(method = "model", lambda = select_best(x = glmnet_tune, metric = "roc_auc")$penalty)

# clean up the data frame for visualization
glmnet_imp |>
  mutate(
    Sign = case_when(
      Sign == "NEG" ~ "More likely from Beyoncé",
      Sign == "POS" ~ "More likely from Taylor Swift"
    ),
    Importance = abs(Importance)
  ) |>
  # importance must be greater than 0
  filter(Importance > 0) |>
  # keep top 20 features for each artist
  slice_max(n = 20, order_by = Importance, by = Sign) |>
  mutate(Variable = fct_reorder(.f = Variable, .x = Importance)) |>
  ggplot(mapping = aes(
    x = Importance,
    y = Variable,
    fill = Sign
  )) +
  geom_col(show.legend = FALSE) +
  scale_fill_brewer(type = "qual") +
  facet_wrap(facets = vars(Sign), scales = "free_y") +
  labs(
    y = NULL,
    title = "Most relevant features for predicting whether\na song is by Beyoncé or Taylor
         Swift",
    subtitle = "Penalized regression model"
  )
```



Session information

This page is built with Quarto.

Cookie Preferences