parsnip

Haojie Liu

2024-01-07

```
essays <- read.csv("train_essays.csv")
prompts <- read.csv("train_prompts.csv")
test <- read.csv("test_essays.csv")
```

word count

```
full_train <- essays %>%
  full_join(prompts)
```

```
## Joining with `by = join_by(prompt_id)`
```

```
full_train <- full_train %>%
  select(-instructions,-source_text, -prompt_name) %>%
  mutate(prompt_id = as.factor(prompt_id),
         generated = as.factor(generated))

unnested_words <- full_train %>%
  unnest_tokens(word, text)

# count the number of words in each essay

word_count <- unnested_words %>%
  group_by(id) %>%
  summarise(word_count = n())

full_train <- full_train %>%
  left_join(word_count)
```

```
## Joining with `by = join_by(id)`
```

```
# we can see that the fake essays usually have a less number of word count

full_train %>%
  group_by(generated) %>%
  summarise(mean(word_count))
```

```
## # A tibble: 2 x 2
##   generated `mean(word_count)`
##   <fct>                  <dbl>
## 1 0                       557.
## 2 1                       261.
```

Sentence count

```r
unnested_sentence <- full_train %>%
  unnest_sentences(sentence, text)

# count the number of words in each essay

sentence_count <- unnested_sentence %>%
  group_by(id) %>%
  summarise(sentence_count = n())

full_train <- full_train %>%
  left_join(sentence_count)
```

```
## Joining with `by = join_by(id)`
```

Vocabulary Richness (Type-Token Ratio):

```r
get_TTR <- function(text) {
    word_list <- str_split(text, "\\W+")[[1]]
    return(length(unique(word_list)) / length(word_list))
}

full_train <- bind_cols(full_train, TTR = sapply(essays$text, get_TTR))
```

Readability Scores:

```r
# Function to count syllables - a very basic approximation
count_syllables <- function(word) {
  syllables <- nchar(gsub("[^aeiouyAEIOUY]", "", word))
  syllables <- ifelse(syllables == 0, 1, syllables) # Ensure at least one syllable
  return(syllables)
}

# Function to calculate readability scores
calculate_readability <- function(text) {
  words <- strsplit(text, "\\s+")[[1]]
  sentences <- strsplit(text, "[.!?]")[[1]]
  total_words <- length(words)
  total_sentences <- length(sentences)
  total_syllables <- sum(sapply(words, count_syllables))

  avg_sentence_length <- total_words / total_sentences
  avg_syllables_per_word <- total_syllables / total_words

  flesch <- 206.835 - 1.015 * avg_sentence_length - 84.6 * avg_syllables_per_word
  flesch_kincaid <- 0.39 * avg_sentence_length + 11.8 * avg_syllables_per_word - 15.59

  return(list(flesch = flesch, flesch_kincaid = flesch_kincaid))
}

# Apply to each essay
readability_scores <- sapply(essays$text, calculate_readability)
```

```r
full_train <- bind_cols(full_train, flesch_reading_ease = readability_scores$flesch, flesch_kincaid_grad
```

```r
full_train <- full_train %>%
  select( -prompt_id) %>%
  select( -text) %>%
  select(-id) %>%
  mutate(generated = as.numeric(generated))
```

```r
convert_test <- function(test){

  # word count

  unnested_words <- test %>%
    unnest_tokens(word, text)

  word_count <- unnested_words %>%
    group_by(id) %>%
    summarise(word_count = n())

  full_test <- test %>%
    left_join(word_count)

  # sentence count

  unnested_sentence <- full_test %>%
    unnest_sentences(sentence, text)

  sentence_count <- unnested_sentence %>%
    group_by(id) %>%
    summarise(sentence_count = n())

  full_test <- full_test %>%
    left_join(sentence_count)

  # Richness score

  full_test <- bind_cols(full_test, TTR = sapply(test$text, get_TTR))

  # Calculate readability scores

  readability_scores <- sapply(essays$text, calculate_readability)

  full_test <- bind_cols(full_test,
                         flesch_reading_ease = readability_scores$flesch,
                         flesch_kincaid_grade = readability_scores$flesch_kincaid)

  full_test <- full_test %>%
    select(-prompt_id) %>%
    select(-id) %>%
    select(-text)

  return(full_test)
}
```

```r
full_test <- convert_test(test)
```

```
## Joining with ‘by = join_by(id)‘
## Joining with ‘by = join_by(id)‘
```

```r
recipe1 <- recipe(generated ~ ., data = full_train)

#model_rf <- rand_forest(
#  trees = tune(),
#  min_n = tune()
#) %>%
#  set_engine("randomForest") %>%
#  set_mode("regression")

#rf_workflow <- workflow() %>%
 # add_model(model_rf) %>%
#  add_recipe(recipe = recipe1)

#train_folds <- vfold_cv(full_train, v=5, strata = 'generated')
#model_param = extract_parameter_set_dials(rf_workflow)

#rf_tune <- rf_workflow %>%
#  tune_grid(train_folds,
 #           grid = model_param %>% grid_random(size = 200))

#autoplot(rf_tune)

#show_best(rf_tune)
```

```r
set.seed(123)

final_model_boost <- boost_tree(
  trees = 1753,
  min_n = 4,
  tree_depth = 13,
  learn_rate = 0.039463909,
  loss_reduction = 2.085301e-10 ,
  sample_size = 0.8635351    ,
  stop_iter = 5
) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

final_boost_workflow <- workflow() %>%
  add_model(final_model_boost) %>%
  add_recipe(recipe = recipe1)


boost_fit <- final_boost_workflow %>%
  fit(full_train)

result <- boost_fit %>%
  predict(full_test)


result <- bind_cols(id = test$id,
                generated = result)
write.csv(result, "submission.csv", row.names=FALSE)

result
```

```
## # A tibble: 3 x 2
##   id        .pred
##   <chr>     <dbl>
## 1 0000aaaa  1.96
## 2 1111bbbb  1.96
## 3 2222cccc  1.96
```