STAT 4601/5703 Tutorial 8

Alex Lehmann

Introduction

In this tutorial, we will employ both supervised and unsupervised learning to classify text by its source. More specifically, we will use topic modeling and a k-nearest neighbors classifier to sort chapters of three classic English-language novels into the correct titles. In the process, we will see applications of the topic modeling theory discussed in last week's tutorial as well as gain exposure to useful programming tools such as regular expressions and revisit some concepts from prior lessons as well.

The topic modeling example below comes is a modified version of an example given in Text Mining with R: A Tidy Approach by Silge and Robinson. See their work for more on the topic.

Setup

```r
library(tidyverse)
library(gutenbergr) # Books!
library(tm) # Text mining objects
library(textstem) # Stemming and lemmatizing utilities
library(topicmodels)
library(tidytext) # Tidy wrangling and cleaning tools for text data
library(tidymodels)
```

Data Acquisition and Preprocessing

The dataset for this tutorial consists of the complete texts of three classic novels: Jules Verne's *Twenty Thousand Leagues under the Sea*, Jane Austen's *Pride and Prejudice*, and Charles Dickens' *Great Expectations*. Fortunately, Project Gutenberg offers a free API to download the complete text of thousands of public-domain books and the `gutenbergr` package provides an R interface to access it.

```
titles <- c("Twenty Thousand Leagues under the Sea",
            "Pride and Prejudice",
            "Great Expectations")
books <- gutenbergr::gutenberg_works(title %in% titles) %>%
  gutenbergr::gutenberg_download(meta_fields = "title")
```

Our task today is to classify the chapters of these books, so let's organize them by chapter. The Gutenberg text contains chapter labels, so we can use a *regular expression* to find each chapter heading and collect the text in between it and the next chapter heading. This lookup occurs in the `stringr::str_detect()` call below. Once we have the chapters separated, we'll break each one down into its individual words in preparation for topic modeling.

```
chapters <- books %>%
  dplyr::group_by(title) %>%
  dplyr::mutate(
    chapter = cumsum(
      stringr::str_detect(
        text, stringr::regex("^chapter ", ignore_case = TRUE)
      )
    )
  ) %>%
  dplyr::ungroup() %>%
  dplyr::filter(chapter > 0) %>%
  tidyr::unite(document, title, chapter)
(chapter_words <- tidytext::unnest_tokens(chapters, word, text))
```

```
# A tibble: 414,716 x 3
   gutenberg_id document                                word
          <int> <chr>                                   <chr>
 1          164 Twenty Thousand Leagues under the Sea_1 chapter
 2          164 Twenty Thousand Leagues under the Sea_1 i
 3          164 Twenty Thousand Leagues under the Sea_1 a
 4          164 Twenty Thousand Leagues under the Sea_1 shifting
 5          164 Twenty Thousand Leagues under the Sea_1 reef
 6          164 Twenty Thousand Leagues under the Sea_1 the
 7          164 Twenty Thousand Leagues under the Sea_1 year
 8          164 Twenty Thousand Leagues under the Sea_1 1866
 9          164 Twenty Thousand Leagues under the Sea_1 was
10          164 Twenty Thousand Leagues under the Sea_1 signalised
```

```
# i 414,706 more rows
```

Regular expressions ("regex") are a formal language for describing patterns in strings of written characters. They allow programmers to specify patterns, or "shapes", of characters rather than the characters themselves. This offers a more flexible way to accomplish tasks like finding all phone numbers or postal codes in a large block of text. The regular expression above, `"^chapter "`, matches every instance of the word "chapter", followed by a space, at the beginning of a line of text - this is how we're finding our chapter delineations.

Now we can total up the counts of each word in each chapter. To save some time, we'll remove the stop words now. Stop words are words like "a", "the", and "and": words which are part of the language's grammatical structure but don't carry much meaning relevant to our task. We'll drop these with `anti_join()` and then count the occurrences of each word in each chapter.

```
word_counts <- chapter_words %>%
  dplyr::anti_join(tidytext::stop_words) %>%
  dplyr::count(document, word, sort = TRUE)
word_counts
```

```
# A tibble: 90,985 x 3
   document               word        n
   <chr>                  <chr>   <int>
 1 Great Expectations_57 joe        88
 2 Great Expectations_7  joe        70
 3 Great Expectations_17 biddy      63
 4 Great Expectations_27 joe        58
 5 Great Expectations_38 estella    58
 6 Great Expectations_2  joe        56
 7 Great Expectations_23 pocket     53
 8 Great Expectations_15 joe        50
 9 Great Expectations_18 joe        50
10 Great Expectations_9  joe        44
# i 90,975 more rows
```

Now we'll convert this count list to a document-term matrix - we're using `tidytext` to accomplish this conversion, but bear in mind that the `DocumentTermMatrix` object comes from the `tm` package.

```
(dtm <- tidytext::cast_dtm(word_counts, document, word, n))
```

```
<<DocumentTermMatrix (documents: 166, terms: 16531)>>
```

```
Non-/sparse entries: 90985/2653161
Sparsity           : 97%
Maximal term length: 19
Weighting          : term frequency (tf)
```

Topic Modeling

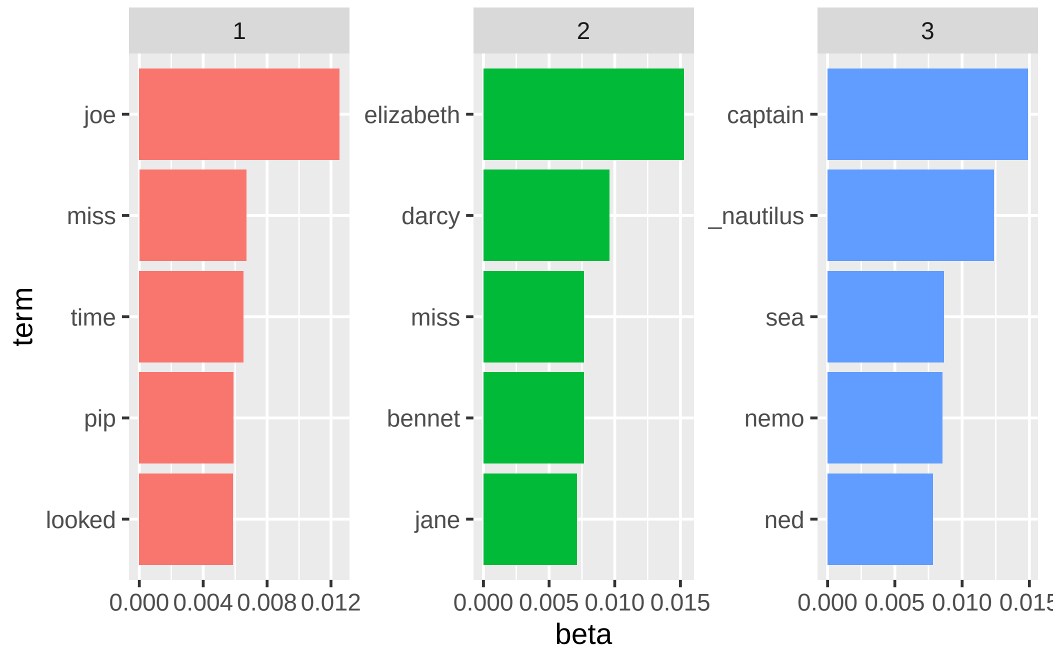
We will now fit a topic model to the chapters of the books to construct a representation of the text that will be easier for our classifier to work with. We'll use latent Dirichlet allocation (LDA), a very popular topic model introduced in 2003. This implementation comes from the `topicmodels` package, which (by default) uses a variational expectation-maximization algorithm to fit a hierarchical Bayesian mixture model to the corpus. We know the texts come from three books, so we can use three topics to get there; in other cases, you'll need to try different numbers of topics and see how things go.

```r
(chapters_lda <- topicmodels::LDA(dtm, k = 3, control = list(seed = 1234)))
```

```
A LDA_VEM topic model with 3 topics.
```

LDA has two main outputs we're interested in: a description of the importance of each word to each topic (β), and a description of the mixture of topics comprising each document (γ). Let's start by seeing which words are most important to which topics.

```r
topics <- tidytext::tidy(chapters_lda, matrix = "beta")
topics %>%
  dplyr::group_by(topic) %>%
  dplyr::slice_max(beta, n = 5) %>%
  dplyr::ungroup() %>%
  dplyr::mutate(term = tidytext::reorder_within(term, beta, topic)) %>%
  ggplot2::ggplot(aes(x = beta, y = term, fill = factor(topic))) +
  ggplot2::geom_col(show.legend = FALSE) +
  ggplot2::facet_wrap(~ topic, scales = "free") +
  tidytext::scale_y_reordered()
```



It looks like LDA learned the different characters involved in the three books. This makes sense: these words are the most likely distinguishing characteristics between the three books. A human reader might use this same tactic to tell the chapters apart.

Now we can look at the mixtures of topics in each book. The approach is largely the same as for the topics themselves.

```
mixtures <- chapters_lda %>%
  tidytext::tidy(matrix = "gamma") %>%
  dplyr::arrange(document)
mixtures
```

```
# A tibble: 498 x 3
   document                topic     gamma
   <chr>                   <int>     <dbl>
 1 Great Expectations_1        1 1.00
 2 Great Expectations_1        2 0.0000410
 3 Great Expectations_1        3 0.0000410
 4 Great Expectations_10       1 1.00
 5 Great Expectations_10       2 0.0000295
 6 Great Expectations_10       3 0.0000295
 7 Great Expectations_11       1 1.00
 8 Great Expectations_11     2 0.0000146
 9 Great Expectations_11     3 0.0000146
10 Great Expectations_12     1 1.00
# i 488 more rows
```

There's some strong polarization between the topics in each chapter. That's another sign that this representation is going to be good for classification.

Classification

With a helpful representation of the chapters in hand, we'll now move on to classification. We should probably start with a train-test split. Note the use of regular expressions to pull the book names out of the chapter labels to make the case labels.

```r
split <- mixtures %>%
  dplyr::mutate(
    topic = paste0("Topic_", topic),
    book = stringr::str_extract(document, ".+(?=_)")
  ) %>%
  tidyr::pivot_wider(names_from = "topic", values_from = "gamma") %>%
  rsample::initial_split(prop = 0.5, strata = book)
train <- rsample::training(split)
test <- rsample::testing(split)
```

Now we'll build our model. Tidymodels supports k-nearest neighbor classification, so we'll use the same methods we covered previously.

```r
knn_model <- parsnip::nearest_neighbor(dist_power = 1) %>%
  parsnip::set_engine("kknn") %>%
  parsnip::set_mode("classification")

knn_recipe <- recipes::recipe(
  book ~ Topic_1 + Topic_2 + Topic_3, data = train
)
```

Fitting the model is the same as well.

```r
knn_wflow <- workflows::workflow() %>%
  workflows::add_model(knn_model) %>%
  workflows::add_recipe(knn_recipe)
knn_fit <- generics::fit(knn_wflow, data = train)
```

Now we'll have our classifier try to figure out which books the test set chapters came from.

```
(predictions <- dplyr::bind_cols(test, predict(knn_fit, test)))
```

```
# A tibble: 84 x 6
   document               book                Topic_1  Topic_2 Topic_3 .pred_class
   <chr>                  <chr>                 <dbl>    <dbl>   <dbl> <fct>
 1 Great Expectations_1   Great Expectations    1.00   4.10e-5 4.10e-5 Great Expe~
 2 Great Expectations_10 Great Expectations    1.00   2.95e-5 2.95e-5 Great Expe~
 3 Great Expectations_14 Great Expectations    0.977  2.26e-2 1.01e-4 Great Expe~
 4 Great Expectations_2   Great Expectations    1.00   2.14e-5 2.14e-5 Great Expe~
 5 Great Expectations_20 Great Expectations    1.00   2.38e-5 2.38e-5 Great Expe~
 6 Great Expectations_22 Great Expectations    0.692  1.91e-1 1.17e-1 Great Expe~
 7 Great Expectations_23 Great Expectations    0.589  1.91e-1 2.20e-1 Great Expe~
 8 Great Expectations_25 Great Expectations    0.988  2.49e-5 1.24e-2 Great Expe~
 9 Great Expectations_26 Great Expectations    1.00   2.73e-5 2.73e-5 Great Expe~
10 Great Expectations_28 Great Expectations    1.00   3.16e-5 3.16e-5 Great Expe~
# i 74 more rows
```

Now let's see how many were correct.

```
predictions %>%
  dplyr::filter(book == .pred_class) %>%
  nrow() %>%
  `/`(nrow(test))
```

```
[1] 1
```

It's a perfect classification! This illustrates just how powerful topic modeling is as a machine learning tool for solving problems involving text data. This is a simple case, though - we only have three classes and there isn't a ton of overlap between the books. Adding additional books would make this problem more interesting - I encourage everyone to do so on your own time. I also encourage you to try finding a topic model for this data with more than three topics. You may be able to find a more detailed representation of these texts and being finding similarities as well as differences.