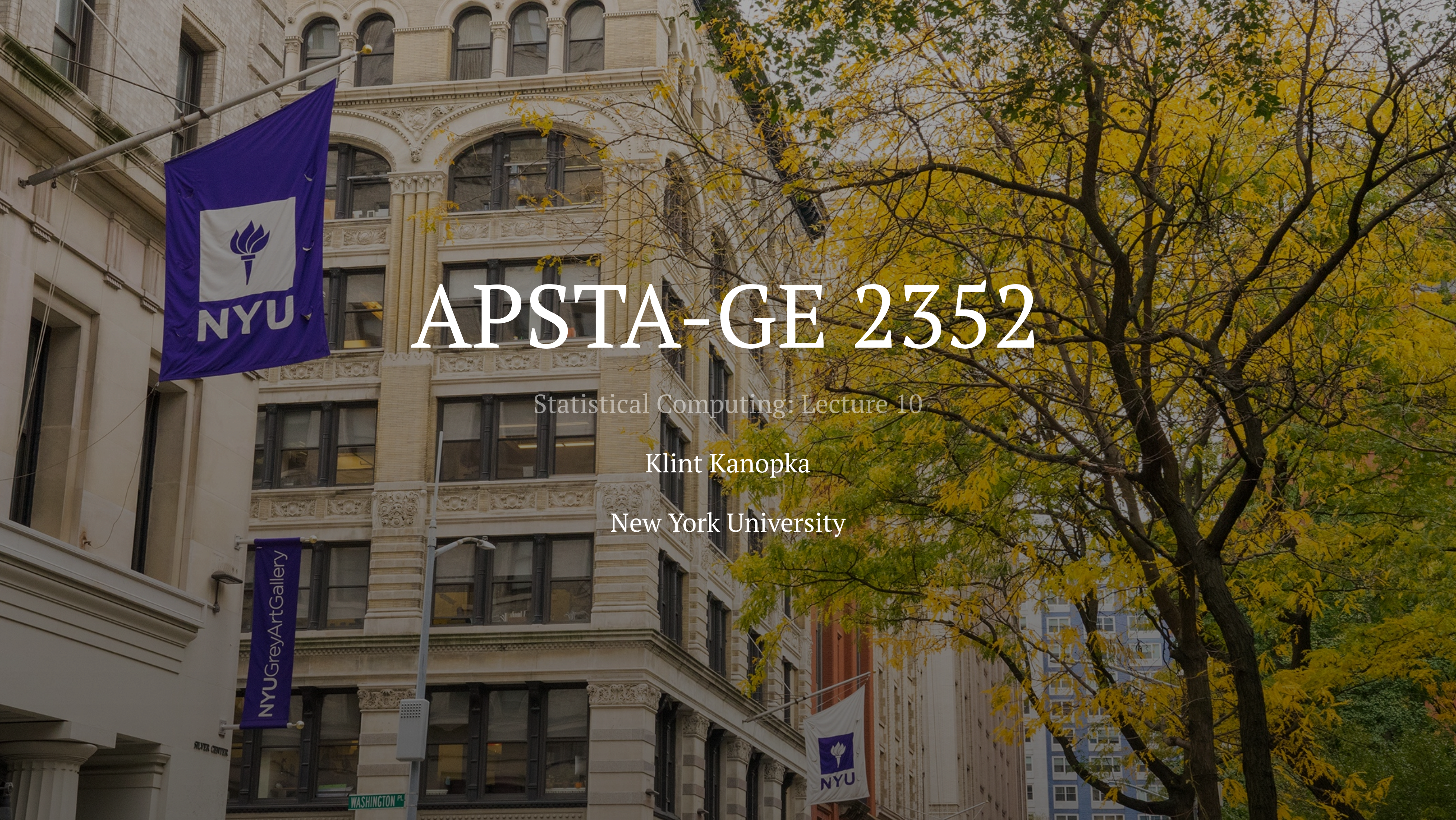


APSTA-GE 2352

Statistical Computing: Lecture 10

Klint Kanopka

New York University

Table of Contents

Check-In

- PollEv.com/klintkanopka

Announcements

- PS5 due in a week
- Upcoming PRIISM seminar talk with my friend Charlie Rahal from Oxford
  - Associate Professor in Data Science and Informatics @ University of Oxford
  - Wednesday, Nov 19 @11a
  - Talk is about algorithmic fairness and the potential consequences of using `set.seed()` incorrectly in your research
  - Check Slack for more info!
- Tomorrow the SCSS PhD students are giving talks
  - 10a in Kimball 301W
  - Should be super interesting, may give you ideas or places you can get involved in research!

A New Problem

Problem

- Today we want to try to understand why people hate movies
- To do this, we'll have a collection of the text of 12,500 **negative** reviews from IMDB
- Our goal is to try to understand what people hate about movies

Matrix Factorization

Matrix Factorization

- Less actual linear algebra than when we talked about PCA!
- Let's first think about factoring: Where have you used it in the past?
- Say we have a $m \times n$ real matrix, $\mathbf{M}$
- We can factor it into: $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$
  - $\mathbf{U}$ is a $m \times m$ real orthogonal matrix and the columns are called the *left singular vectors*
  - $\mathbf{V}$ is a $n \times n$ real orthogonal matrix and the columns are called the *right singular vectors*
  - Σ is a $m \times n$ positive diagonal rectangular matrix (aka: $\sigma_i = \Sigma_{ii}$ and $\Sigma_{i \neq j} = 0$), and the diagonal entries (called *singular values*) are arranged in decreasing order
  - We can write the whole original matrix as $\mathbf{M} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$
- This is the Singular Value Decomposition (SVD). Sometimes SVD refers to the *compact SVD*, where Σ is an $r \times r$ matrix that only contains the r non-zero singular values, and $\mathbf{U}$ is $m \times r$ and $\mathbf{V}$ is $n \times r$
  - The compact SVD just throws away a bunch of zeros

Singular Value Decomposition

- Does the SVD remind you of anything?
- As it turns out, the SVD of a data matrix is how PCA is usually computed
  - This is because of a relationship between the SVD and the eigenvalues of $\mathbf{M}^\top \mathbf{M}$
- Note that the SVD recovers our original matrix *exactly*! It is not an approximation!
- But, the SVD requires a lot of computational time for large matrices and sometimes we only want the first few (aka most important) singular vectors
- We introduce now the *truncated SVD*
  - The truncated SVD is the same setup as the compact SVD, with one modification:
  - We only keep the left and right singular vectors associated with the first r singular values!
  - This means the truncated SVD **is** an approximation if $r < \mathbf{rank}(\mathbf{M})$

Truncated SVD

- The truncated SVD is *much* faster to compute than the full SVD
- Done in a method similar to power iteration
- The truncated SVD is written as: $\tilde{\mathbf{M}} = \mathbf{U}\Sigma\mathbf{V}^\top$
- The truncated SVD minimizes the *Frobenius Norm*: $||\mathbf{M} - \tilde{\mathbf{M}}||_F$
- Guess what the Frobenius Norm is

Truncated SVD

- The truncated SVD is *much* faster to compute than the full SVD
- Done in a method similar to power iteration
- The truncated SVD is written as: $\tilde{\mathbf{M}} = \mathbf{U}\Sigma\mathbf{V}^\top$
- The truncated SVD minimizes the *Frobenius Norm*: $||\mathbf{M} - \tilde{\mathbf{M}}||_F$
- Guess what the Frobenius Norm is

$$||A||_F = \sqrt{\sum_i^m \sum_j^n a_{ij}^2}$$

Truncated SVD

- The truncated SVD is *much* faster to compute than the full SVD
- Done in a method similar to power iteration
- The truncated SVD is written as: $\tilde{\mathbf{M}} = \mathbf{U}\Sigma\mathbf{V}^\top$
- The truncated SVD minimizes the *Frobenius Norm*: $||\mathbf{M} - \tilde{\mathbf{M}}||_F$
- Guess what the Frobenius Norm is

$$||A||_F = \sqrt{\sum_i^m \sum_j^n a_{ij}^2}$$

- This means that the result of the truncated SVD, $\tilde{\mathbf{M}}$, is the best rank r approximation of the original matrix, $\mathbf{M}$

More Similarity Metrics

- How do you know that two things are similar?
- In some ways, "similarity" is the inverse of "distance"
- You can use distance formulas (Euclidean, Manhattan)
- Minkowsky Distance generalizes the two:

$$D(A,B) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

More Similarity Metrics

- For vectors, we frequently use *cosine similarity*
  - Measures the angle between two vectors
  - Cares about the *direction* they point in, but not the *length*

$$S_c(A, B) = \frac{\mathbf{A} \cdot \mathbf{B}}{||A||||B||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}}$$

Similarity between other things

- For sets, we often use Jaccard Similarity
  - The number of items that are common to both sets divided by the number of items across both sets
  - Very common with binary (or dichotomous) data!

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Similarity between other strings

- For strings of text, there are a ton of options:
  - Hamming Distance counts the number of positions where the two strings are different
    - `hamburger` and `burgers` have a Hamming distance of 8
    - `here` and `there` have a Hamming distance of 5
  - Levenshtein distance measures the minimum number of single-character edits required to change one into the other
    - Counts insertions, deletions, and substitutions
    - `hamburger` and `burgers` have a Levenshtein distance of 4
    - `here` and `there` have a Levenshtein distance of 1

The Quantification of Text

Quick text vocabulary

- A *token* is an individual unit of text (usually a word)
- A *document* is a collection of tokens
- A *corpus* is a collection of documents
- A *vocabulary* is the collection of tokens represented in a corpus

Quantification of text

- All of the statistics and models we use to describe things act on *numbers*
- How might we turn text into numbers?

Quantification of text

- All of the statistics and models we use to describe things act on *numbers*
- How might we turn text into numbers?
- The options boil down to basically two things:
  1. Count words (classical NLP)
  2. Embed words in a vector space (deep NLP)

Quantification of text

- All of the statistics and models we use to describe things act on *numbers*
- How might we turn text into numbers?
- The options boil down to basically two things:
  1. Count words (classical NLP)
  2. Embed words in a vector space (deep NLP)
- From here, we aggregate up to the document level in some way

Quantification of text

- All of the statistics and models we use to describe things act on *numbers*
- How might we turn text into numbers?
- The options boil down to basically two things:
  1. Count words (classical NLP)
  2. Embed words in a vector space (deep NLP)
- From here, we aggregate up to the document level in some way
- The most common classical way to represent a corpus for analysis is the *Document Term Matrix* (DTM)
  - One row per document in a corpus
  - One column per word in the vocab
  - Word counts in the individual cells
  - Treats documents as a *bag of words*

Preprocessing

- Raw text is kind of a mess!

Preprocessing

- Raw text is kind of a mess!
- At the most basic level, we want to strip punctuation and convert words to lowercase
    - What other problems can arise from just counting words?

Preprocessing

- Raw text is kind of a mess!
- At the most basic level, we want to strip punctuation and convert words to lowercase
  - What other problems can arise from just counting words?
- **Stop words** are super common words
  - e.g., a, an, the
  - Typically non-informative about the actual content of text, so we remove them

Preprocessing

- Raw text is kind of a mess!
- At the most basic level, we want to strip punctuation and convert words to lowercase
  - What other problems can arise from just counting words?
- **Stop words** are super common words
  - e.g., a, an, the
  - Typically non-informative about the actual content of text, so we remove them
- Sometimes the "same" word can appear in a different form
  - Singular and plural words (dog and dogs)
  - Adjectives and adverbs (quick, quicker, quickly)
  - **Stemming** is the process of stripping tokens down to stems so that these differences are counted together

Preprocessing

- Raw text is kind of a mess!
- At the most basic level, we want to strip punctuation and convert words to lowercase
  - What other problems can arise from just counting words?
- **Stop words** are super common words
  - e.g., a, an, the
  - Typically non-informative about the actual content of text, so we remove them
- Sometimes the "same" word can appear in a different form
  - Singular and plural words (dog and dogs)
  - Adjectives and adverbs (quick, quicker, quickly)
  - **Stemming** is the process of stripping tokens down to stems so that these differences are counted together
- More important in classical methods
- Not needed for deep methods
- Often you’ll have domain-specific words you also want to drop

Classical Text Analysis

Classical text analysis

- Any thoughts on what we could do with a DTM to learn about the collection of documents?
- We can look at the most common tokens in the corpus
- We can look at the most common tokens in a document, or *term frequency*

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

- We can look at how many documents a word appears in, or *inverse document frequency*

$$\mathrm{idf}(t, D) = \ln \frac{N}{d \in D : t \in D}$$

- A common way to weight is tf-idf:

$$\text{tf-idf}(t, d) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

Dictionary methods

- Dictionaries are collections of words that have been coded to have some particular meaning
  - Conceptually: specific words are associated with ideas, themes, or emotions
  - If you use more of those words, you're expressing more of that thing
- The most common collection of dictionaries is Linguistic Inquiry and Word Count (LIWC)
  - Has dictionaries for affect, emotion, social behavior, references to health and culture, and a ton of other stuff
  - Also counts mechanical properties of text construction
  - These are a *great* place to start with your own data!
  - LIWC costs $129.95 for an academic three year license, but free alternatives exist
- You can also develop your own dictionaries for specific purposes
  - This can be really time consuming to do well
  - Take a look at *The Development and Psychometric Properties of LIWC-22* technical report for more information on how to do this

Topic Models

- Core idea: Topic models are basically principal component analysis performed on the DTM
- A singular value decomposition (SVD) on the DTM is called Latent Semantic Analysis (LSA)
  - This is functionally the same as doing a PCA on the DTM
  - Interpretation and use is identical - each "topic" is a dimension and words are more or less associated with each
- A better topic model is the Latent Dirichlet Allocation (LDA)
  - A Bayesian model (if you don’t know what this means it doesn’t super matter)
  - Structured as such:
    - A *document* is made up of a mixture of *topics*
    - A *topic* is made up of a mixture of *words*
    - Here, *mixture* means probability distribution

Topic Models

- Dealing with topic models is conceptually similar to dealing with PCA
- The interpretation you did in lab yesterday is the move!
- There are lots of types of topic models
    - Correlated topic models
    - Structural topic models
        - Allow for topic mixtures to vary as a function of other covariates
        - Use the excellent `stm` package for these

Natural Language Processing with Deep Learning

Word vectors

- The more modern approach is to not preprocess text, but instead *embed* it in a high dimensional vector space
- This means instead of word counts, documents are a sequence of high dimensional (typically $N \approx 300$) vectors
- Word2Vec was the classic implementation
  - "You shall know a word by the company it keeps"
  - Words that co-occur are pushed closer together in the vector space
  - Words that don’t co-occur are spread farther apart
  - Typically trained on an enormous corpus (Google News, Google Books, etc)
- Word vectors capture latent features of language usage
  - See Garg, et al. (2018) for the most famous word embedding paper
- Word vectors can also be aggregated in different ways to get up to document-level embeddings

Transformer Models

- Transformers are a specific deep learning architecture developed by Google scientists
  - Vaswani, A., et al. (2017). Attention is all you need. *Advances in neural information processing systems, 30.*
- Bidirectional Encoder Representations from Transformers (BERT) provide contextual embeddings that allow words to have different representations based on context
- Seq2seq models map sequences of embeddings to other sequences of embeddings and drove most machine translation
- Large Language Models (LLMs) like ChatGPT are wildly over-parameterized transformer models that work on vector embeddings

Language Models

What is a language model?

- In general: a model that describes how text is *generated*
- Models should be *generative*, meaning that they can produce new text
- Models should also be *evaluative*, meaning they can quantify the degree to which something looks like "real" text
- This is, primarily (but not always), done by next-token-prediction
- Tons of ways to do this some work better than others
  - Transformer based models generally work best
  - Recurrent neural network models (RNNs) were the old state of the art
  - Before that, models were purely statistical and based on word and/or character frequency
- Part 3 of ps5 builds a character based language model—let’s talk about what that is and how it works

A character-based language model

- **Core Idea:** Some character combinations are common (i.e., `th`, `on`, `er`) and some are not (i.e., `qz`, `xx`, `uu`), so we have some idea of what English words *should* look like
- If we train a model from some corpus by counting the instances of two letter pairs, we can compare two pieces new text and see which set of character combinations looks more likely to exist given our trained model

Building the language model

- Beginning from our training text:
  1. Preprocess the text down to just individual letter information
  2. Count pairs of characters: `teeth` gets a count for `te` , `ee` , `et` , and `th`
  3. Put all of these counts into a table where you can retrieve them easily
  4. Turn all of these counts into log probabilities

Using the language model

1. Take a string of text and separate it into character pairs:

- `onions` becomes `on`, `ni`, `io`, `on`, `ns`

2. Look up each character pair in your log probability table
3. Add the log probabilities of each character pair together to get the log probability of the entire string
4. Real words should contain observed pairs of characters, and therefore be considered more likely (higher probability)

Latent Semantic Analysis

- How could we try to find themes in text using computational methods?
- **Setup:** I have a bunch of negative movie reviews, and I want to extract similar themes to understand why people hate movies

Latent Semantic Analysis

- How could we try to find themes in text using computational methods?
- **Setup:** I have a bunch of negative movie reviews, and I want to extract similar themes to understand why people hate movies
- First we need a plan to quantify the text

Latent Semantic Analysis

- How could we try to find themes in text using computational methods?
- **Setup:** I have a bunch of negative movie reviews, and I want to extract similar themes to understand why people hate movies
- First we need a plan to quantify the text
- Then what can we do to the text?

Latent Semantic Analysis

- How could we try to find themes in text using computational methods?
- **Setup:** I have a bunch of negative movie reviews, and I want to extract similar themes to understand why people hate movies
- First we need a plan to quantify the text
- Then what can we do to the text?
- **Latent Semantic Analysis** is the SVD of the document-term matrix

Latent Semantic Analysis

- How could we try to find themes in text using computational methods?
- **Setup:** I have a bunch of negative movie reviews, and I want to extract similar themes to understand why people hate movies
- First we need a plan to quantify the text
- Then what can we do to the text?
- **Latent Semantic Analysis** is the SVD of the document-term matrix
- Think of this as doing "PCA on word counts"

Exploring our text

```r
imdb_neg_raw <- read_csv('data/neg-imdb.csv') |>
  select(review = FileName, text = Content)

imdb_neg_raw$text[2]
```

An example review:

```
Well...tremors I, the original started off in 1990 and i found the movie quite
enjoyable to watch. however, they proceeded to make tremors II and III. Trust
me, those movies started going downhill right after they finished the first one,
i mean, ass blasters??? Now, only God himself is capable of answering the
question \"why in Gods name would they create another one of these dumpster
dives of a movie?\" Tremors IV cannot be considered a bad movie, in fact it
cannot be even considered an epitome of a bad movie, for it lives up to more
than that. As i attempted to sit though it, i noticed that my eyes started to
bleed, and i hoped profusely that the little girl from the ring would crawl
through the TV and kill me. did they really think that dressing the people who
had stared in the other movies up as though they we're from the wild west would
make the movie (with the exact same occurrences) any better? honestly, i would
never suggest buying this movie, i mean, there are cheaper ways to find things
that burn well.
```

Transforming the data into word counts

```r
imdb_neg <- imdb_neg_raw |>
  unnest_tokens(word, text)

imdb_neg
```

Transforming the data into word counts

```r
imdb_neg <- imdb_neg_raw |>
  unnest_tokens(word, text)

imdb_neg

#   review     word
#   <chr>      <chr>
# 1 1821_4.txt working
# 2 1821_4.txt with
# 3 1821_4.txt one
# 4 1821_4.txt of
# 5 1821_4.txt the
```

Transforming the data into word counts

```r
imdb_neg <- imdb_neg_raw |>
  unnest_tokens(word, text)

imdb_neg

#   review     word
#   <chr>      <chr>
# 1 1821_4.txt working
# 2 1821_4.txt with
# 3 1821_4.txt one
# 4 1821_4.txt of
# 5 1821_4.txt the

count(imdb_neg, word, sort = TRUE)
```

Transforming the data into word counts

```r
imdb_neg <- imdb_neg_raw |>
  unnest_tokens(word, text)

imdb_neg

#   review     word
#   <chr>      <chr>
# 1 1821_4.txt working
# 2 1821_4.txt with
# 3 1821_4.txt one
# 4 1821_4.txt of
# 5 1821_4.txt the

count(imdb_neg, word, sort = TRUE)

#   word       n
#   <chr>  <int>
# 1 the   163174
# 2 a      79225
# 3 and    74347
# 4 of     68999
# 5 to     68969
```

Removing stop words

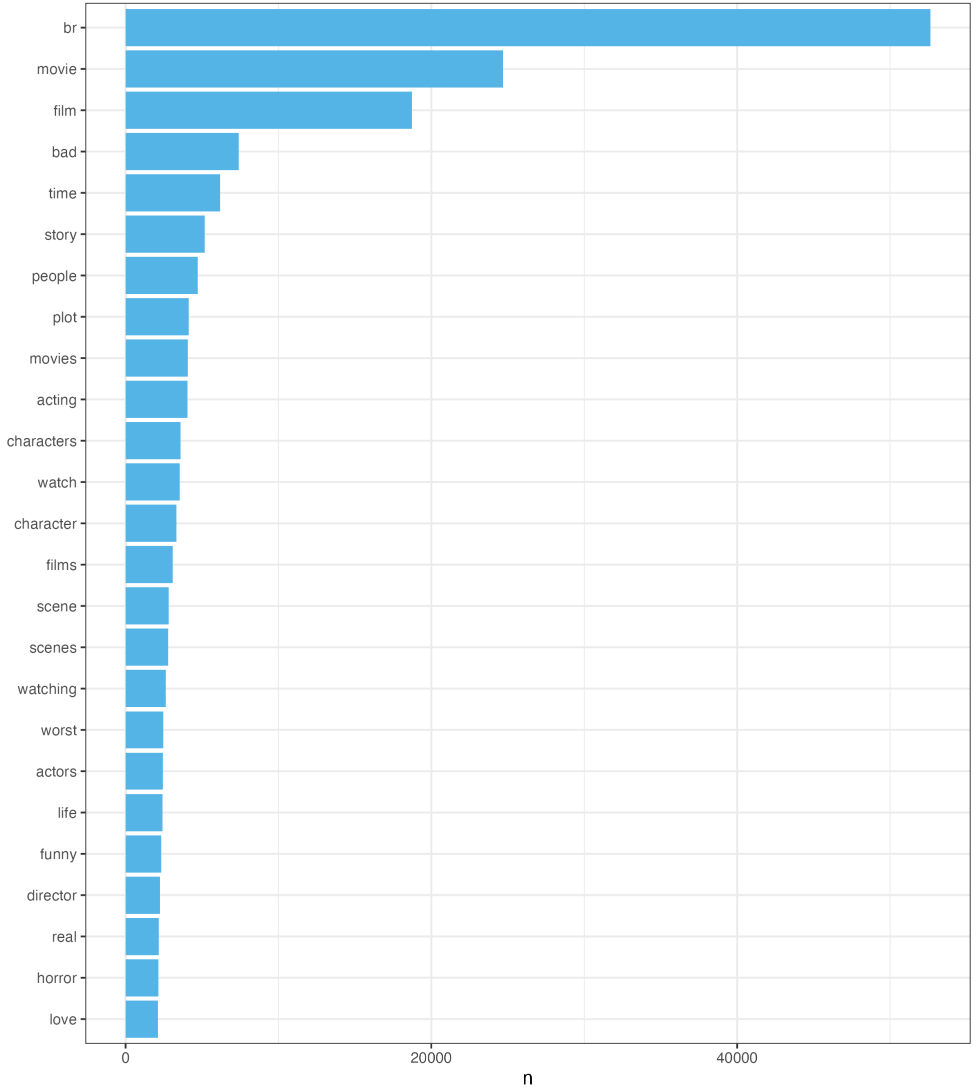
```r
data(stop_words)
stop_words

imdb_neg <- imdb_neg |>
  anti_join(stop_words)

imdb_neg
count(imdb_neg, word, sort = TRUE)
```

Removing stop words

```r
data(stop_words)
stop_words

imdb_neg <- imdb_neg |>
  anti_join(stop_words)

imdb_neg
count(imdb_neg, word, sort = TRUE)

#     word        n
#    <chr>   <int>
#  1 br      52636
#  2 movie   24698
#  3 film    18717
#  4 bad      7389
#  5 time     6200
```

Visualizing most common words

```r
imdb_neg |>
  count(word, sort = TRUE) |>
  head(25) |>
  ggplot(aes(x = n, y = reorder(word, n))) +
  geom_col(fill = okabeito_colors(2)) +
  labs(y = NULL) +
  theme_bw()
```



Count by reviews and drop uninformative words

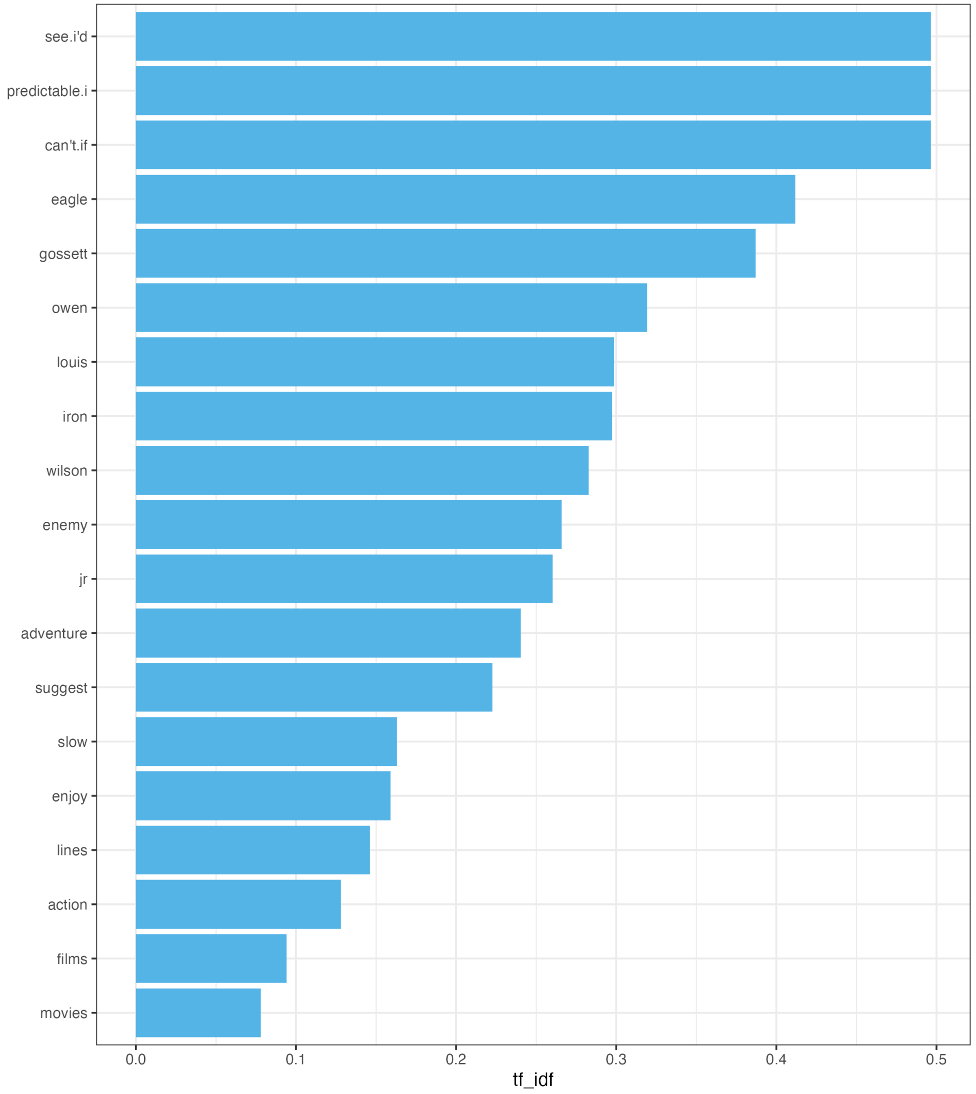
```r
imdb_counts <- imdb_neg |>
  group_by(review, word) |>
  summarize(n = n(), .groups = 'drop')

imdb_counts <- imdb_counts |>
  filter(!word %in% c('br', 'movie', 'film'))
```

Count by reviews and drop uninformative words

```r
imdb_counts <- imdb_neg |>
  group_by(review, word) |>
  summarize(n = n(), .groups = 'drop')

imdb_counts <- imdb_counts |>
  filter(!word %in% c('br', 'movie', 'film'))

imdb_counts

#     review   word               n
#    <chr>    <chr>           <int>
# 1 0_3.txt absurd              2
# 2 0_3.txt audience            1
# 3 0_3.txt briefly             1
# 4 0_3.txt chantings           1
# 5 0_3.txt cinematography      1
```

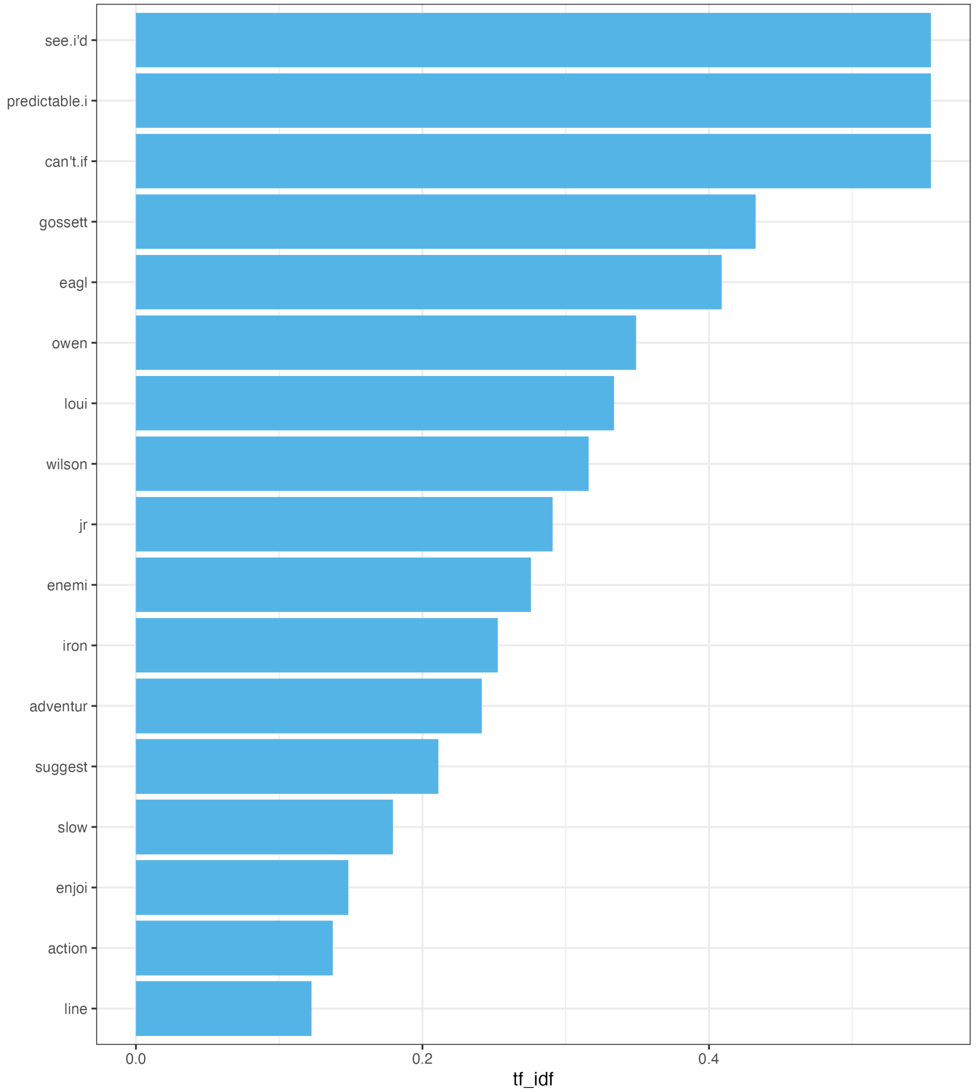
tf-idf weighting

```r
tf_idf <- bind_tf_idf(imdb_counts,
                      word,
                      review,
                      n)

target <- sample(unique(imdb_neg$review), 1)

tf_idf |>
  filter(review == target) |>
  arrange(tf_idf) |>
  head(25) |>
  ggplot(aes(x = tf_idf,
             y = reorder(word, tf_idf))) +
  geom_col(fill = okabeito_colors(2)) +
  labs(y = NULL) +
  theme_bw()
```



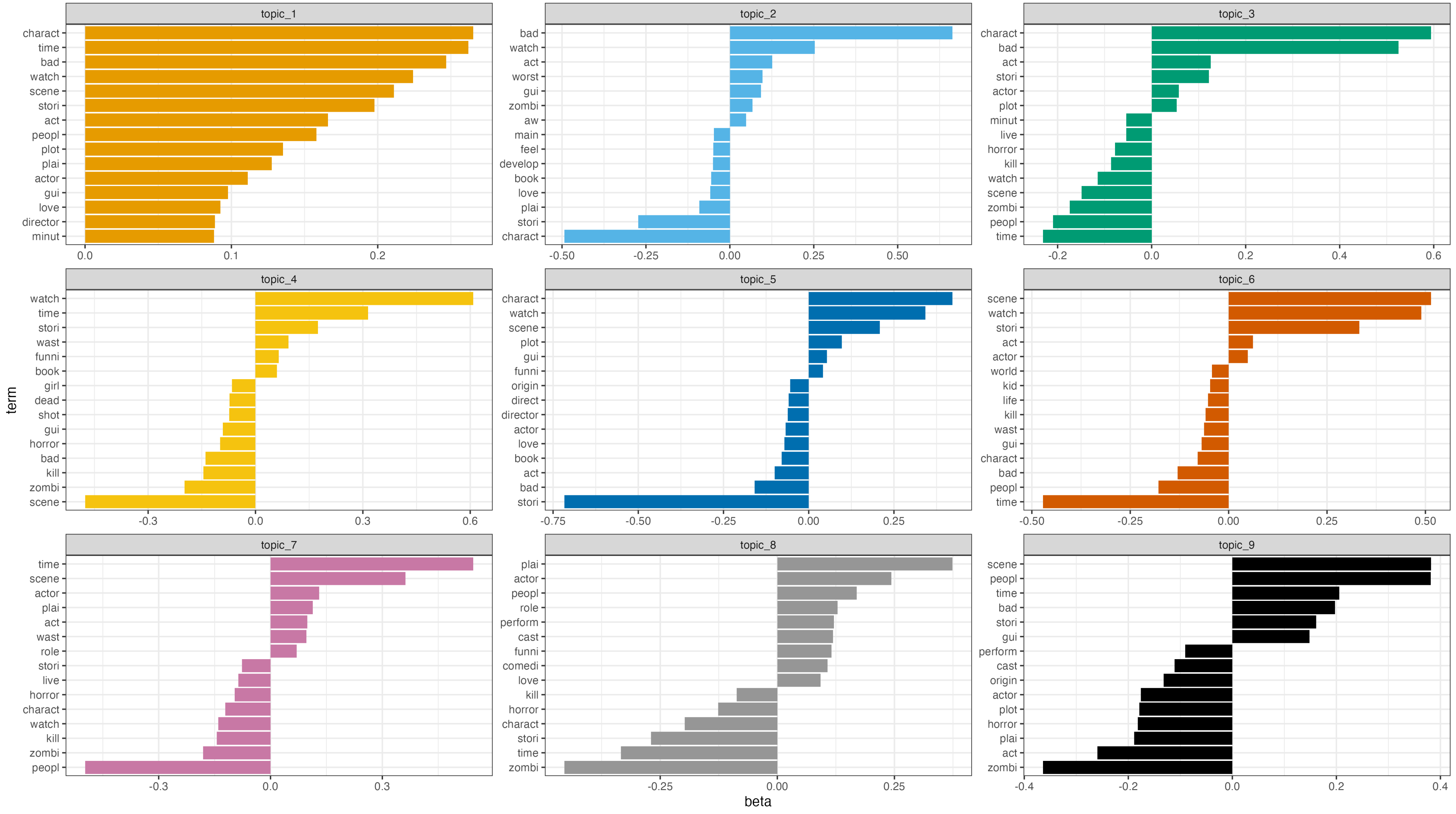
Stemming

```r
imdb_neg <- imdb_neg |>
  mutate(stem = wordStem(word))

imdb_counts <- imdb_neg |>
  group_by(review, stem) |>
  summarize(n = n(), .groups = 'drop') |>
  filter(!stem %in% c('br', 'movi', 'film'))

tf_idf <- bind_tf_idf(imdb_counts,
                      stem,
                      review,
                      n)

tf_idf |>
  filter(review == target) |>
  arrange(tf_idf) |>
  head(25) |>
  ggplot(aes(x = tf_idf,
             y = reorder(stem, tf_idf))) +
  geom_col(fill = okabeito_colors(2)) +
  labs(y = NULL) +
  theme_bw()
```



Construct DTM, conduct LSA, describe topics

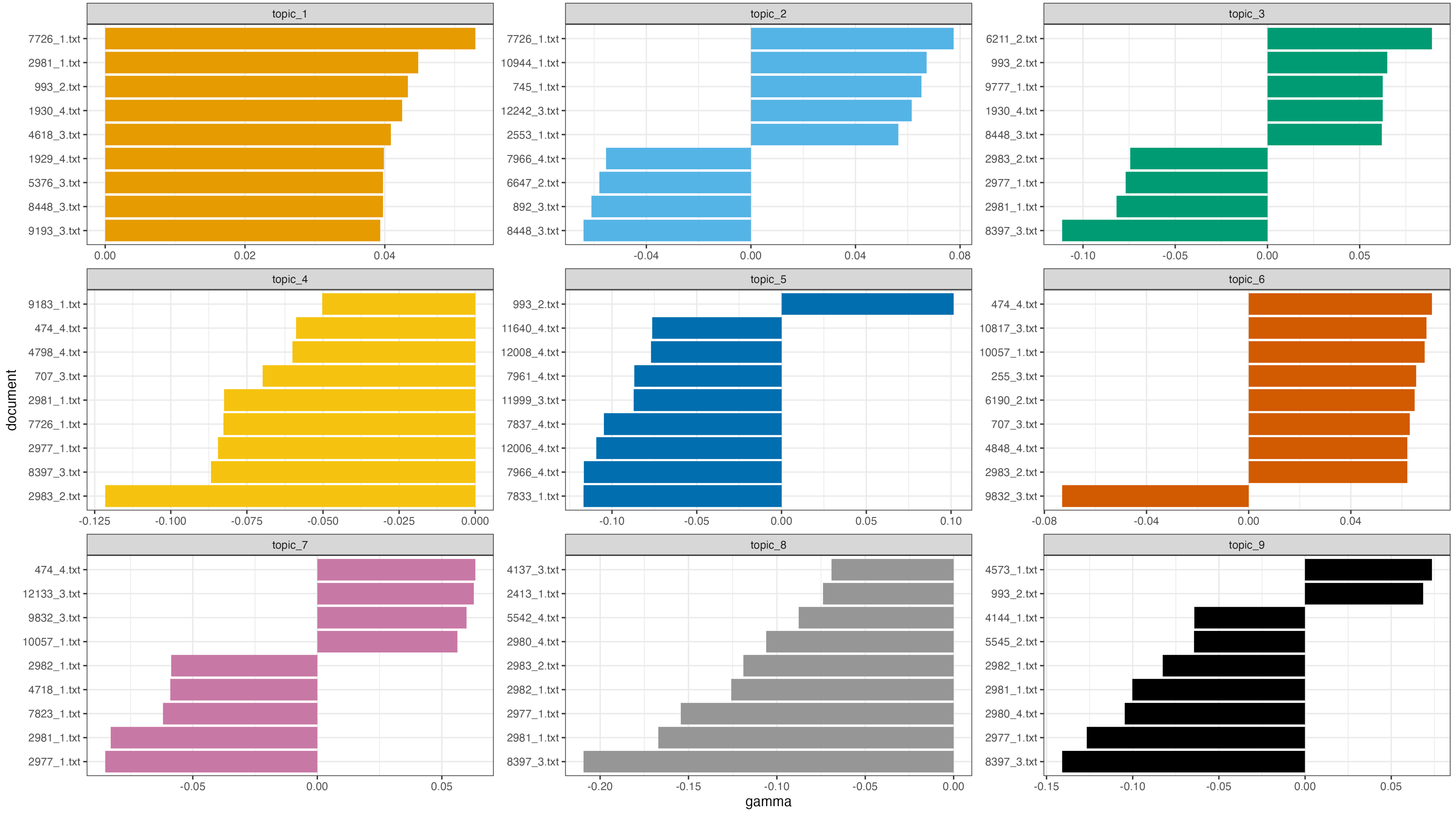
```r
dtm <- cast_dtm(imdb_counts, review, stem, n)

n_topics <- 9
imdb_lsa <- svds(as.matrix(dtm), n_topics)

imdb_topics <- as.data.frame(imdb_lsa$v)
colnames(imdb_topics) <- paste0('topic_', 1:n_topics)
imdb_topics$term <- colnames(dtm)

imdb_top_terms <- imdb_topics |>
  pivot_longer(-term, names_to = 'topic', values_to = 'beta') |>
  group_by(topic) |>
  slice_max(abs(beta), n = 15) |>
  ungroup() |>
  arrange(topic, -beta)

imdb_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) |>
  ggplot(aes(x = beta, y = term, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```



topic_1
charact
time
bad
watch
scene
stori
act
peopl
plot
plai
actor
gui
love
director
minut
0.0
0.1
0.2
topic_2
bad
watch
act
worst
gui
zombi
aw
main
feel
develop
book
love
plai
stori
charact
-0.50
-0.25
0.00
0.25
0.50
topic_3
charact
bad
act
stori
actor
plot
minut
live
horror
kill
watch
scene
zombi
peopl
time
-0.2
0.0
0.2
0.4
0.6
topic_4
watch
time
stori
wast
funni
book
girl
dead
shot
gui
horror
bad
kill
zombi
scene
-0.3
0.0
0.3
0.6
topic_5
charact
watch
scene
plot
gui
funni
origin
direct
director
actor
love
book
act
bad
stori
-0.75
-0.50
-0.25
0.00
0.25
topic_6
scene
watch
stori
act
actor
world
kid
life
kill
wast
gui
charact
bad
peopl
time
-0.50
-0.25
0.00
0.25
0.50
topic_7
time
scene
actor
plai
act
wast
role
stori
live
horror
charact
watch
kill
zombi
peopl
-0.3
0.0
0.3
topic_8
plai
actor
peopl
role
perform
cast
funni
comedi
love
kill
horror
charact
stori
time
zombi
-0.25
0.00
0.25
topic_9
scene
peopl
time
bad
stori
gui
perform
cast
origin
actor
plot
horror
plai
act
zombi
-0.4
-0.2
0.0
0.2
0.4
term
beta

Most prominent reviews by topic

```r
imdb_reviews <- as.data.frame(imdb_lsa$u)
colnames(imdb_reviews) <- paste0('topic_', 1:n_topics)
imdb_reviews$document <- rownames(dtm)

imdb_top_reviews <- imdb_reviews |>
  pivot_longer(-document, names_to = 'topic', values_to = 'gamma') |>
  group_by(topic) |>
  slice_max(abs(gamma), n = 9) |>
  ungroup() |>
  arrange(topic, -gamma)

imdb_top_reviews |>
  mutate(document = reorder_within(document, gamma, topic)) |>
  ggplot(aes(x = gamma, y = document, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```



topic_1
7726_1.txt
2981_1.txt
993_2.txt
1930_4.txt
4618_3.txt
1929_4.txt
5376_3.txt
8448_3.txt
9193_3.txt
0.00
0.02
0.04
topic_2
7726_1.txt
10944_1.txt
745_1.txt
12242_3.txt
2553_1.txt
7966_4.txt
6647_2.txt
892_3.txt
8448_3.txt
-0.04
0.00
0.04
0.08
topic_3
6211_2.txt
993_2.txt
9777_1.txt
1930_4.txt
8448_3.txt
2983_2.txt
2977_1.txt
2981_1.txt
8397_3.txt
-0.10
-0.05
0.00
0.05
topic_4
9183_1.txt
474_4.txt
4798_4.txt
707_3.txt
2981_1.txt
7726_1.txt
2977_1.txt
8397_3.txt
2983_2.txt
-0.125
-0.100
-0.075
-0.050
-0.025
0.000
topic_5
993_2.txt
11640_4.txt
12008_4.txt
7961_4.txt
11999_3.txt
7837_4.txt
12006_4.txt
7966_4.txt
7833_1.txt
-0.10
-0.05
0.00
0.05
0.10
topic_6
474_4.txt
10817_3.txt
10057_1.txt
255_3.txt
6190_2.txt
707_3.txt
4848_4.txt
2983_2.txt
9832_3.txt
-0.08
-0.04
0.00
0.04
topic_7
474_4.txt
12133_3.txt
9832_3.txt
10057_1.txt
2982_1.txt
4718_1.txt
7823_1.txt
2981_1.txt
2977_1.txt
-0.05
0.00
0.05
topic_8
4137_3.txt
2413_1.txt
5542_4.txt
2980_4.txt
2983_2.txt
2982_1.txt
2977_1.txt
2981_1.txt
8397_3.txt
-0.20
-0.15
-0.10
-0.05
0.00
topic_9
4573_1.txt
993_2.txt
4144_1.txt
5545_2.txt
2982_1.txt
2981_1.txt
2980_4.txt
2977_1.txt
8397_3.txt
-0.15
-0.10
-0.05
0.00
0.05
document
gamma

Use cosine similarity between right singular vectors to find similar reviews

```r
U <- imdb_lsa$u
rownames(U) <- rownames(dtm)

norm <- sqrt(matrix(rowSums(U * U), byrow = F, nrow = nrow(U), ncol = nrow(U)))
cos_sim <- U %*% t(U) / (norm * t(norm))

cos_sim_df <- as.data.frame(cos_sim) |>
  rownames_to_column('review_1') |>
  pivot_longer(-review_1, names_to = 'review_2', values_to = 'cos_sim') |>
  filter(review_1 != review_2) |>
  filter(cos_sim < 0.98) |>
  arrange(-cos_sim)

head(cos_sim_df, 5)

#    review_1    review_2    cos_sim
#   <chr>       <chr>          <dbl>
# 1 10223_4.txt 11884_4.txt    0.980
# 2 11884_4.txt 10223_4.txt    0.980
# 3 2297_4.txt  8846_2.txt     0.980
# 4 8846_2.txt  2297_4.txt     0.980
# 5 2505_2.txt  8018_1.txt     0.980
```

Let’s look at an example:

```r
imdb_neg_raw |>
  filter(review %in% c('10223_4.txt', '11884_4.txt')) |>
  pull(text)
```

```
[1] "The worlds largest inside joke. The world's largest, most exclusive inside joke.<br /><br />Emulating the
brash and 'everyman' humor of office space, this film drives the appeal of this film into the ground by making
the humor such that it would only be properly appreciated by legal secretaries writing books. The audience is
asked to assume the unfamiliar role of a legal secretary, and then empathize with the excruciatingly dumb..."

[2] "I remembered the title so well. To me, it was a Flora Robson movie with Olivier and Vivien Leigh in
supporting roles. And it had Vincent Massey's voice from behind whiskers. Well Flora Robson was great. Her next
signature, for me, would be \"55 Days at Peking\". The same role but with different sumptuous gowns. And the
same voice. As for the Armada, it was a subtext. I like black-and-white films. Was everything done in..."
```

Let’s filter out the common words:

```r
filter_words <- tf_idf |>
  select(stem, idf) |>
  distinct() |>
  filter(idf < 2.5)
filtered_imdb_counts <- anti_join(imdb_counts, filter_words, by = 'stem')
filtered_dtm <- cast_dtm(filtered_imdb_counts, review, stem, n)
f_imdb_lsa <- svds(as.matrix(filtered_dtm), n_topics)
f_imdb_topics <- as.data.frame(f_imdb_lsa$v)
colnames(f_imdb_topics) <- paste0('topic_', 1:n_topics)
f_imdb_topics$term <- colnames(filtered_dtm)
f_imdb_top_terms <- f_imdb_topics |>
  pivot_longer(-term, names_to = 'topic', values_to = 'beta') |>
  group_by(topic) |>
  slice_max(abs(beta), n = 10) |>
  ungroup() |>
  arrange(topic, -beta)
f_imdb_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) |>
  ggplot(aes(x = beta, y = term, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```

Let’s filter out the common words:

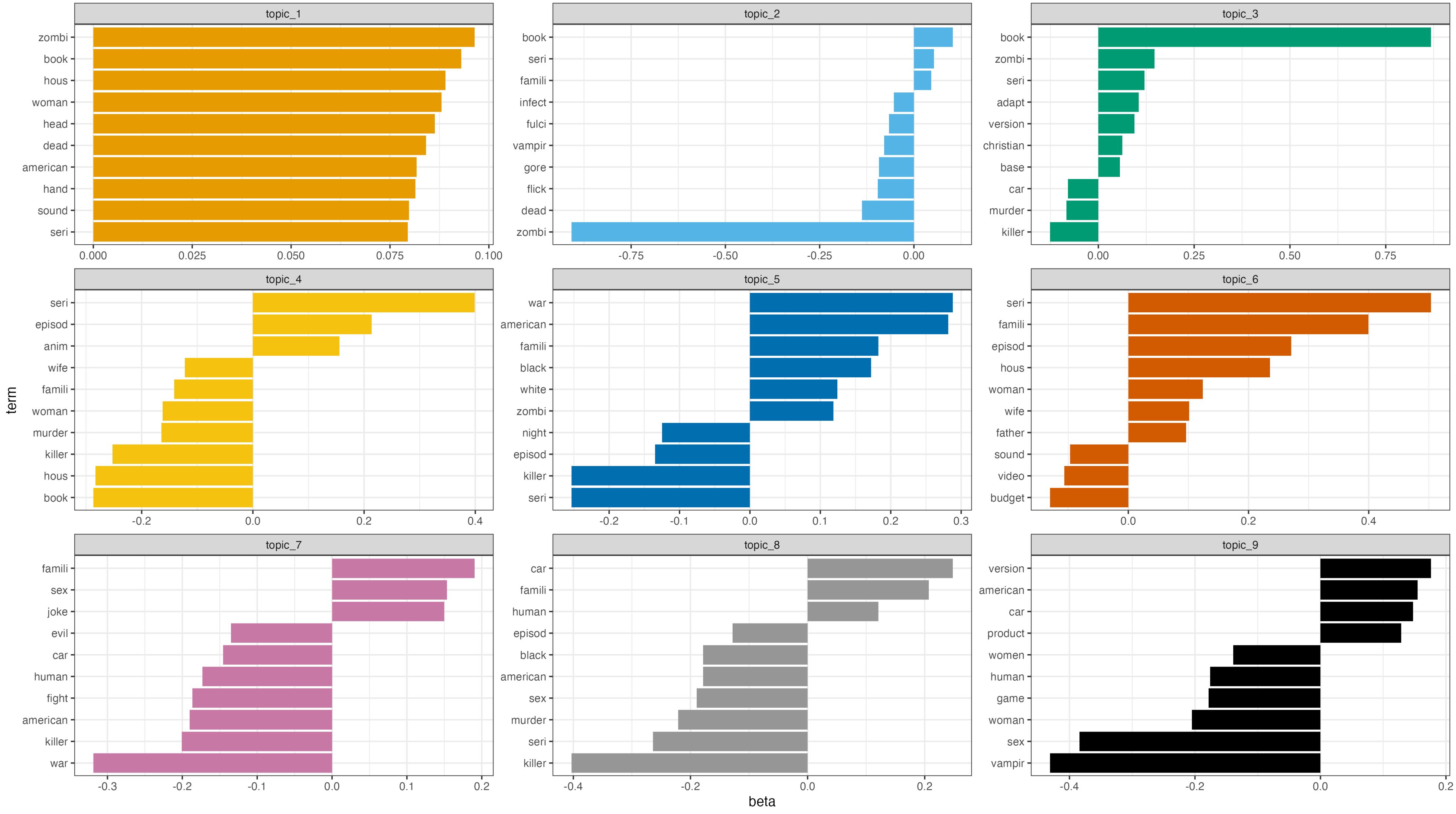
```r
filter_words <- tf_idf |>
  select(stem, idf) |>
  distinct() |>
  filter(idf < 2.5)
filtered_imdb_counts <- anti_join(imdb_counts, filter_words, by = 'stem')
filtered_dtm <- cast_dtm(filtered_imdb_counts, review, stem, n)
f_imdb_lsa <- svds(as.matrix(filtered_dtm), n_topics)
f_imdb_topics <- as.data.frame(f_imdb_lsa$v)
colnames(f_imdb_topics) <- paste0('topic_', 1:n_topics)
f_imdb_topics$term <- colnames(filtered_dtm)
f_imdb_top_terms <- f_imdb_topics |>
  pivot_longer(-term, names_to = 'topic', values_to = 'beta') |>
  group_by(topic) |>
  slice_max(abs(beta), n = 10) |>
  ungroup() |>
  arrange(topic, -beta)
f_imdb_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) |>
  ggplot(aes(x = beta, y = term, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```

Let’s filter out the common words:

```r
filter_words <- tf_idf |>
  select(stem, idf) |>
  distinct() |>
  filter(idf < 2.5)
filtered_imdb_counts <- anti_join(imdb_counts, filter_words, by = 'stem')
filtered_dtm <- cast_dtm(filtered_imdb_counts, review, stem, n)
f_imdb_lsa <- svds(as.matrix(filtered_dtm), n_topics)
f_imdb_topics <- as.data.frame(f_imdb_lsa$v)
colnames(f_imdb_topics) <- paste0('topic_', 1:n_topics)
f_imdb_topics$term <- colnames(filtered_dtm)
f_imdb_top_terms <- f_imdb_topics |>
  pivot_longer(-term, names_to = 'topic', values_to = 'beta') |>
  group_by(topic) |>
  slice_max(abs(beta), n = 10) |>
  ungroup() |>
  arrange(topic, -beta)
f_imdb_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) |>
  ggplot(aes(x = beta, y = term, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```

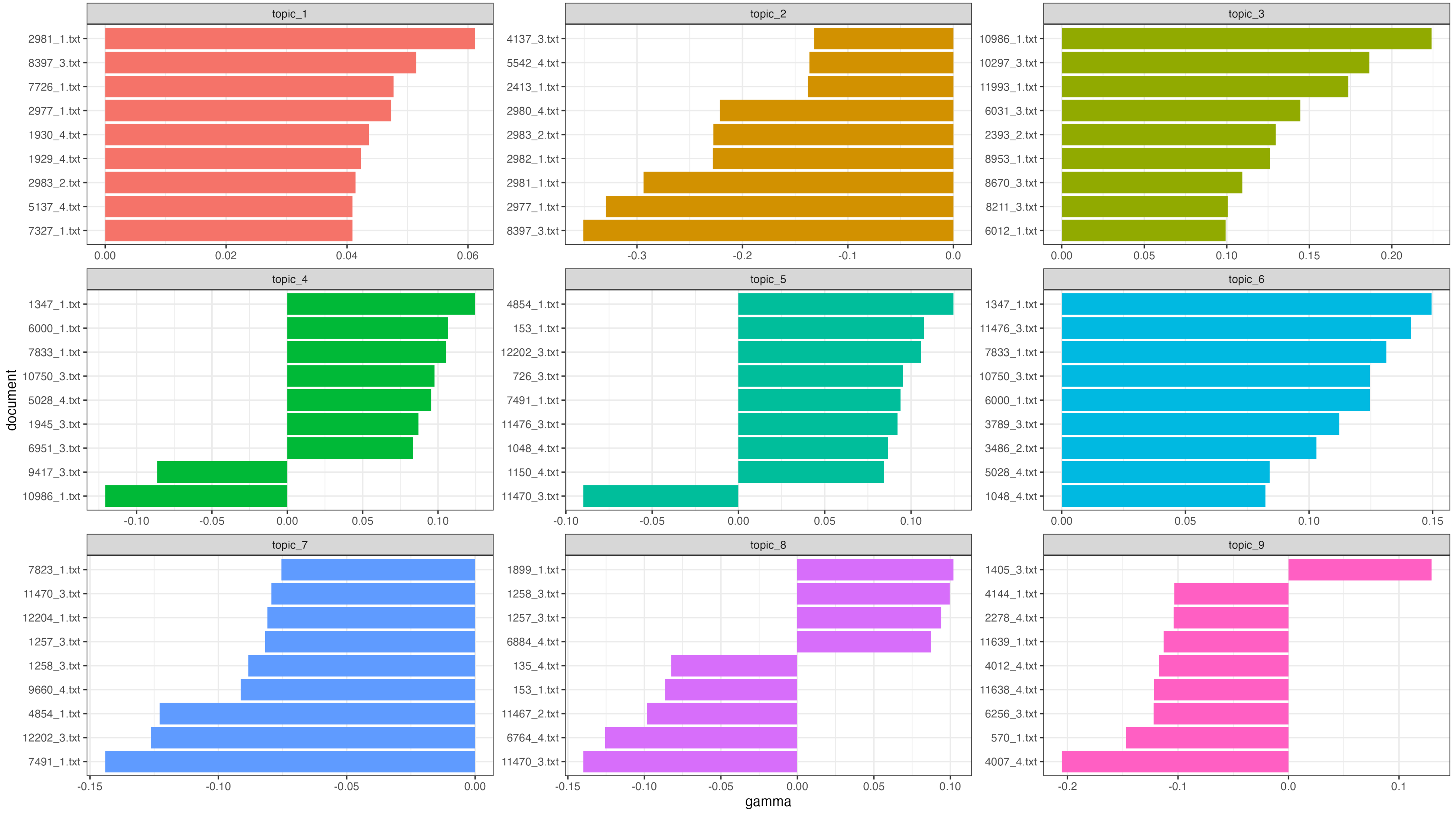
Let’s filter out the common words:

```r
filter_words <- tf_idf |>
  select(stem, idf) |>
  distinct() |>
  filter(idf < 2.5)
filtered_imdb_counts <- anti_join(imdb_counts, filter_words, by = 'stem')
filtered_dtm <- cast_dtm(filtered_imdb_counts, review, stem, n)
f_imdb_lsa <- svds(as.matrix(filtered_dtm), n_topics)
f_imdb_topics <- as.data.frame(f_imdb_lsa$v)
colnames(f_imdb_topics) <- paste0('topic_', 1:n_topics)
f_imdb_topics$term <- colnames(filtered_dtm)
f_imdb_top_terms <- f_imdb_topics |>
  pivot_longer(-term, names_to = 'topic', values_to = 'beta') |>
  group_by(topic) |>
  slice_max(abs(beta), n = 10) |>
  ungroup() |>
  arrange(topic, -beta)
f_imdb_top_terms |>
  mutate(term = reorder_within(term, beta, topic)) |>
  ggplot(aes(x = beta, y = term, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_fill_okabeito() +
  scale_y_reordered() +
  theme_bw()
```


topic_1
zombi
book
hous
woman
head
dead
american
hand
sound
seri
0.000
0.025
0.050
0.075
0.100
topic_2
book
seri
famili
infect
fulci
vampir
gore
flick
dead
zombi
-0.75
-0.50
-0.25
0.00
topic_3
book
zombi
seri
adapt
version
christian
base
car
murder
killer
0.00
0.25
0.50
0.75
topic_4
seri
episod
anim
wife
famili
woman
murder
killer
hous
book
-0.2
0.0
0.2
0.4
topic_5
war
american
famili
black
white
zombi
night
episod
killer
seri
-0.2
-0.1
0.0
0.1
0.2
0.3
topic_6
seri
famili
episod
hous
woman
wife
father
sound
video
budget
0.0
0.2
0.4
topic_7
famili
sex
joke
evil
car
human
fight
american
killer
war
-0.3
-0.2
-0.1
0.0
0.1
0.2
topic_8
car
famili
human
episod
black
american
sex
murder
seri
killer
-0.4
-0.2
0.0
0.2
topic_9
version
american
car
product
women
human
game
woman
sex
vampir
-0.4
-0.2
0.0
0.2
term
beta

Top reviews by topic

```r
f_imdb_reviews <- as.data.frame(f_imdb_lsa$u)
colnames(f_imdb_reviews) <- paste0('topic_', 1:n_topics)
f_imdb_reviews$document <- rownames(filtered_dtm)

f_imdb_top_reviews <- f_imdb_reviews |>
  pivot_longer(-document, names_to = 'topic', values_to = 'gamma') |>
  group_by(topic) |>
  slice_max(abs(gamma), n = 9) |>
  ungroup() |>
  arrange(topic, -gamma)

f_imdb_top_reviews |>
  mutate(document = reorder_within(document, gamma, topic)) |>
  ggplot(aes(x = gamma, y = document, fill = as.character(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(. ~ topic, scales = 'free') +
  scale_y_reordered() +
  theme_bw()
```


topic_1
2981_1.txt
8397_3.txt
7726_1.txt
2977_1.txt
1930_4.txt
1929_4.txt
2983_2.txt
5137_4.txt
7327_1.txt
0.00
0.02
0.04
0.06
topic_2
4137_3.txt
5542_4.txt
2413_1.txt
2980_4.txt
2983_2.txt
2982_1.txt
2981_1.txt
2977_1.txt
8397_3.txt
-0.3
-0.2
-0.1
0.0
topic_3
10986_1.txt
10297_3.txt
11993_1.txt
6031_3.txt
2393_2.txt
8953_1.txt
8670_3.txt
8211_3.txt
6012_1.txt
0.00
0.05
0.10
0.15
0.20
topic_4
1347_1.txt
6000_1.txt
7833_1.txt
10750_3.txt
5028_4.txt
1945_3.txt
6951_3.txt
9417_3.txt
10986_1.txt
-0.10
-0.05
0.00
0.05
0.10
topic_5
4854_1.txt
153_1.txt
12202_3.txt
726_3.txt
7491_1.txt
11476_3.txt
1048_4.txt
1150_4.txt
11470_3.txt
-0.10
-0.05
0.00
0.05
0.10
topic_6
1347_1.txt
11476_3.txt
7833_1.txt
10750_3.txt
6000_1.txt
3789_3.txt
3486_2.txt
5028_4.txt
1048_4.txt
0.00
0.05
0.10
0.15
topic_7
7823_1.txt
11470_3.txt
12204_1.txt
1257_3.txt
1258_3.txt
9660_4.txt
4854_1.txt
12202_3.txt
7491_1.txt
-0.15
-0.10
-0.05
0.00
topic_8
1899_1.txt
1258_3.txt
1257_3.txt
6884_4.txt
135_4.txt
153_1.txt
11467_2.txt
6764_4.txt
11470_3.txt
-0.15
-0.10
-0.05
0.00
0.05
0.10
topic_9
1405_3.txt
4144_1.txt
2278_4.txt
11639_1.txt
4012_4.txt
11638_4.txt
6256_3.txt
570_1.txt
4007_4.txt
-0.2
-0.1
0.0
0.1
document
gamma

Example texts

- Zombie topic:

```r
imdb_neg_raw |>
  filter(review %in% c('8397_3.txt') |>
  pull(text)
```

```
[1] "After reading the previous comments, I'm just glad that I wasn't the only person left confused, especially
by the last 20 minutes. John Carradine is shown twice walking down into a grave and pulling the lid shut after
him. I anxiously awaited some kind of explanation for this odd behavior...naturally I assumed he had something
to do with the evil goings-on at the house, but since he got killed off by the first rising corpse (hereafter
referred to as Zombie #1)...
```

- Book topic:

```r
imdb_neg_raw |>
  filter(review %in% c('10986_1.txt')) |>
  pull(text)
```

```
[1] "When I saw that Mary Louise Parker was associated with this epic novel turned film, I was intrigued. Being
a fan of the book, I assumed she'd be playing Tony, Roz, or Charis, but more so, I was intrigued to see how they
would turn this very head-y, almost psychological (but not psychological thriller) novel in to a movie that
would be accessible to those who hadn't read the novel, and that would be at least mildly satisfying for those
who had. The book is a complex reflection of society, women, and modern life..."
```

Wrap Up

Recap

- Matrix factorization, via the SVD, shows up everywhere!
- It’s a great way to break up more complicated data into easier to understand parts
- The ways in which we quantify text limit (or augment) how we can learn from it
- Most classical methods are *bag of words* models that just count instances of tokens
- Topic models function like PCA on the document term matrix
- Modern deep learning methods use vector embeddings of words to carry more information in each token

Final Thoughts

- PollEv.com/klintkanopka