Universität
Konstanz

Studying News Use with Computational Methods

Text Analysis in R, Part II: Topic Modeling

Julian Unkel
University of Konstanz
2021/06/28

Agenda

Universität Konstanz

Especially with large text corpora, we may want to use methods to explore the textual content and discover meaningful patterns. Unsupervised machine learning methods structure text corpora into latent classes without much user input.

Topic modeling describes one family of methods to uncover such meaningful patterns in large text corpora.

Our agenda today:

- Topic models
  - Basics
  - Model fitting
  - Model selection
  - Model interpretation
  - Adding covariates
- Keyword-assisted topic models
  - Defining a-priori topics
  - Model fitting
  - Model selection
  - Model interpretation
- Validating topic models

Universität Konstanz

Topic models

Topic models

Topic models describe a family of similar methods to uncover meaningful patterns in documents based on their textual content. Among the most common methods are *LDA* (Latent Dirichlet Allocation), *CTM* (Correlated Topic Models), and *STM* (Structural Topic Models).

Topic models

All methods share some common assumptions:

- Text corpora consist of D documents (e.g. news articles, social media posts; individual documents numbered $d_1, d_2, \ldots$) and V terms (i.e., words; individual terms numbered $w_1, w_2, \ldots$). Documents can be represented as bags-of-words.
- Text corpora can be represented by K latent topics which sit hierarchically between the whole corpus and invidiual documents. Each document d_i and each word w_i may "belong" with differing probabilities to topic $k_1, k_2, \ldots$ (mixed membership). K has to be set by the researcher.
- We want to estimate the matrices $D \times K$ and $V \times K$ which contain the document probabilites per topic, and the word probabilities per topic, respectively.
- This is achieved by modeling a data generating process that describes the creation of documents as first drawing a probability distribution of topics for each document $d_1, d_2, \ldots$. For each word in document d_i, we then draw a topic from the document's topic distribution, and then a word from the topic's word distribution.

Topic models

- The word-topic matrix $V \times K$ may then be used to describe and interpret meaning of topics $k_1, k_2, \ldots$, for example by looking at the words with the highest conditional probability for topic k_j.
- The document-topic matrix $D \times K$ may be used to assign documents to topics, for example by assinging each document d_i to topic k_j with the highest conditional probability.
- Different topic modeling procedures differ mainly by the probability distributions used to represent the topic probabilities.

In this class, we will use *STM* (Structural Topic Modeling) for ease of use and the ability to add covariates.

Setup

Setup as usual:

```r
library(tidyverse)
library(tidytext)
library(quanteda)
library(stm)
```

Setup

Load the Guardian corpus. As last time, we also create a variable for the day the article was published.

```
guardian_tibble <- readRDS("data/guardian_sample_2020.rds") %>%
    mutate(day = lubridate::date(date))
```

Preprocess as usual:

```
guardian_corpus <- corpus(guardian_tibble,
                          docid_field = "id", text_field = "body")

guardian_tokens <- guardian_corpus %>%
  tokens(remove_punct = TRUE, remove_symbols = TRUE, remove_numbers = TRUE,
         remove_url = TRUE, remove_separators = TRUE) %>%
  tokens_tolower()

guardian_dfm <- guardian_tokens %>%
  dfm()
```

Setup

DFM trimming may affect the outcome of topic modeling quite strongly. We usually want to remove common wiords with little discriminating value and very short documents to make the topic modeling results more interpretable and reduce computational load:

```
trimmed_dfm <- guardian_dfm %>%
  dfm_trim(max_docfreq = 0.6, min_docfreq = .01, docfreq_type = "prop") %>%
  dfm_remove(stopwords("en", source = "nltk")) %>%
  dfm_subset(ntoken(guardian_dfm) > 5)
```

Topic modeling with stm

We need to convert the DFM to a format suitable for the stm package:

```
stm_dfm <- convert(trimmed_dfm, to = "stm")
str(stm_dfm, max.level = 1)
```

```
## List of 3
##  $ documents:List of 9965
##  $ vocab    : chr [1:5165] "100m" "1950s" "1960s" "1970s" ...
##  $ meta     :'data.frame':    9965 obs. of  5 variables:
```

Model fitting

To fit models, we simply use the `stm()` function. We need to provide the `documents` and the `vocab`, which are both accessible in the `stm_dfm` object. We also need to set the K parameter. We begin by estimating 20 topics (note that this may take quite a long time - use `verbose = TRUE` the follow the progress in the console; as topic models are initialized randomly, it may be useful to also set a `seed` to create reproducible results):

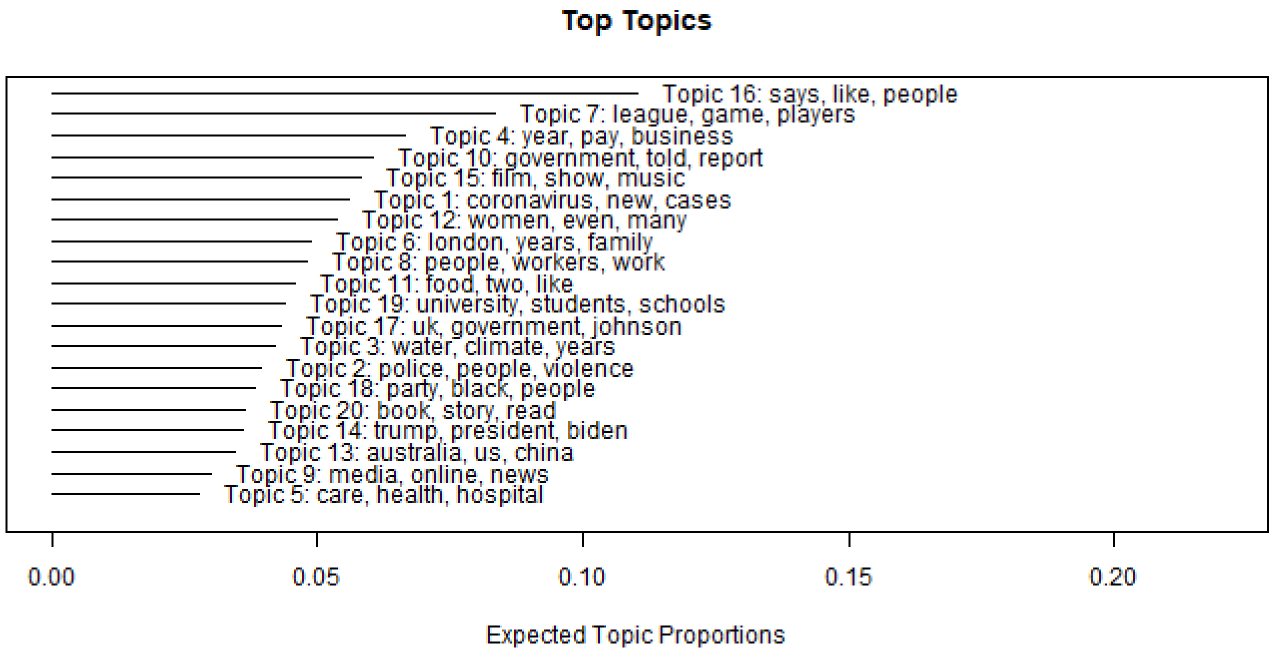
```
guardian_stm_20 <- stm(documents = stm_dfm$documents,
                       vocab = stm_dfm$vocab,
                       K = 20)
guardian_stm_20
```

```
## A topic model with 20 topics, 9965 documents and a 5165 word dictionary.
```

Model fitting

We can use `plot()` and `summary()` functions on the output:

```
plot(guardian_stm_20)
```



Model fitting

```
summary(guardian_stm_20)
```

```
## A topic model with 20 topics, 9965 documents and a 5165 word dictionary.

## Topic 1 Top Words:
##       Highest Prob: coronavirus, new, cases, people, virus, lockdown, covid-19
##       FREX: restrictions, cases, travel, quarantine, outbreak, infections, coronavirus
##       Lift: bridge, gatherings, passengers, quarantine, travellers, cruise, restrictions
##       Score: bridge, cases, coronavirus, virus, infections, restrictions, lockdown
## Topic 2 Top Words:
##       Highest Prob: police, people, violence, officers, two, man, prison
##       FREX: police, officers, prison, violence, protesters, crime, arrested
##       Lift: en, sentenced, custody, police, prison, protesters, officers
##       Score: police, en, officers, violence, protesters, arrested, prison
## Topic 3 Top Words:
##       Highest Prob: water, climate, years, new, year, fire, air
##       FREX: species, environmental, animals, wildlife, land, fires, birds
##       Lift: wildlife, grey, species, conservation, birds, pollution, fires
##       Score: grey, species, climate, wildlife, water, pollution, conservation
## Topic 4 Top Words:
##       Highest Prob: year, pay, business, money, financial, government, economy
##       FREX: income, tax, financial, scheme, debt, unemployment, pay
```

Model selection

Before we start interpreting, we need to talk about setting K. Apart from theoretical considerations, we may use measures such as *semantic coherence* and *exclusivity* to gauge the validity of topic models.

- *Semantic coherence* increases with more words with high topic probabilities appearing in the same documents. Manual intepretation and labelling of topics is usually easier for topics with higher semantic coherence.
- *Exclusivity* increases with more words with high probabilites for one topic having lower probabilites for other topics.
- Both measures usually represent a trade-off: Semantic coherence can be increased simply by estimating fewer topics; exclusivity usually increases with more topics.

Model selection

Compute semantic coherence with `semanticCoherence()`:

```
semanticCoherence(guardian_stm_20, stm_dfm$documents)
```

```
##  [1] -52.60617 -68.15450 -80.08364 -54.77585 -61.35769 -61.44610 -63.25772
##  [8] -45.84327 -90.86718 -62.00340 -62.46925 -45.84828 -74.30635 -45.59453
## [15] -77.92583 -39.70473 -66.85216 -76.90637 -81.31434 -64.50726
```

Model selection

Compute semantic coherence with `exclusivity()`:

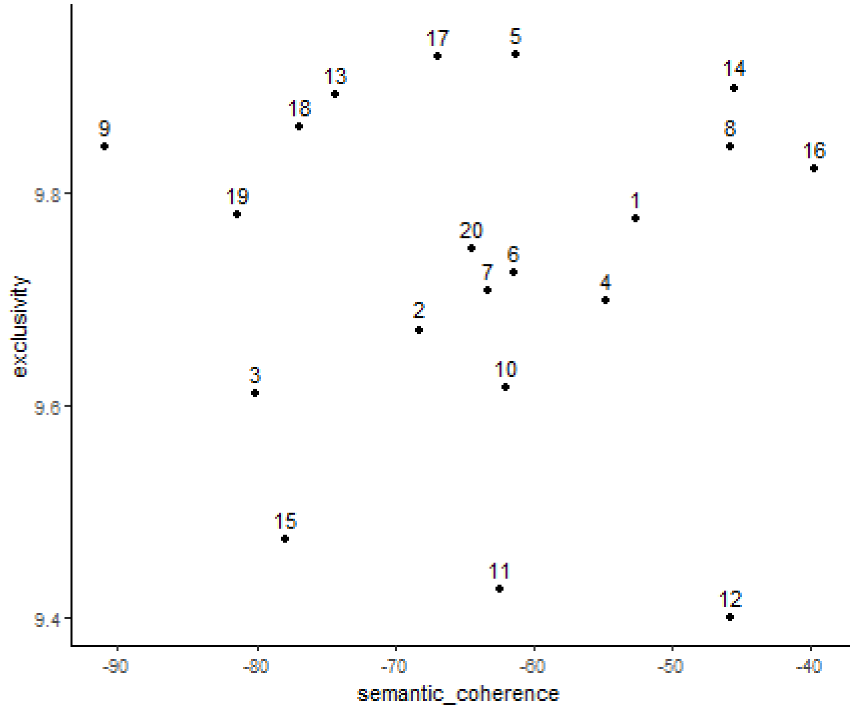
```
exclusivity(guardian_stm_20)
```

```
##  [1] 9.775630 9.670680 9.611312 9.698116 9.930043 9.724218 9.708318 9.842582
##  [9] 9.842910 9.617226 9.426971 9.400283 9.891755 9.899018 9.474194 9.822790
## [17] 9.928742 9.862438 9.779013 9.746267
```

Model selection

To investigate the common trade-off between semantic coherence and exclusivity, it is useful to plot both measures:

```
tibble(
  topic = 1:20,
  exclusivity = exclusivity(guardian_stm_20),
  semantic_coherence = semanticCoherence(guar
  ) %>%
  ggplot(aes(semantic_coherence, exclusivity,
  geom_point() +
  geom_text(nudge_y = .02) +
  theme_classic()
```



Model selection

We can use semantic coherence and exclusivity to compare topic models with a different number K of topics. However, to do so, we must actually fit all models we want to compare.

As this may take some time, it is useful to employ parallelization to speed up the process. Using the `furrr` package, we parallelize model estimation, so depending on the number of available cores, fitting multiple models may actually on take marginally more time than fitting a single model:

```
library(furrr)
plan(multisession)

guardian_models <- tibble(K = c(20, 30, 40, 50, 60)) %>%
  mutate(topic_model = future_map(K, ~stm(documents = stm_dfm$documents,
                                          vocab = stm_dfm$vocab,
                                          K = .,
                                          verbose = FALSE)))
```

Model selection

We can then map the semantic coherence and exclusivity computations on the estimated models:

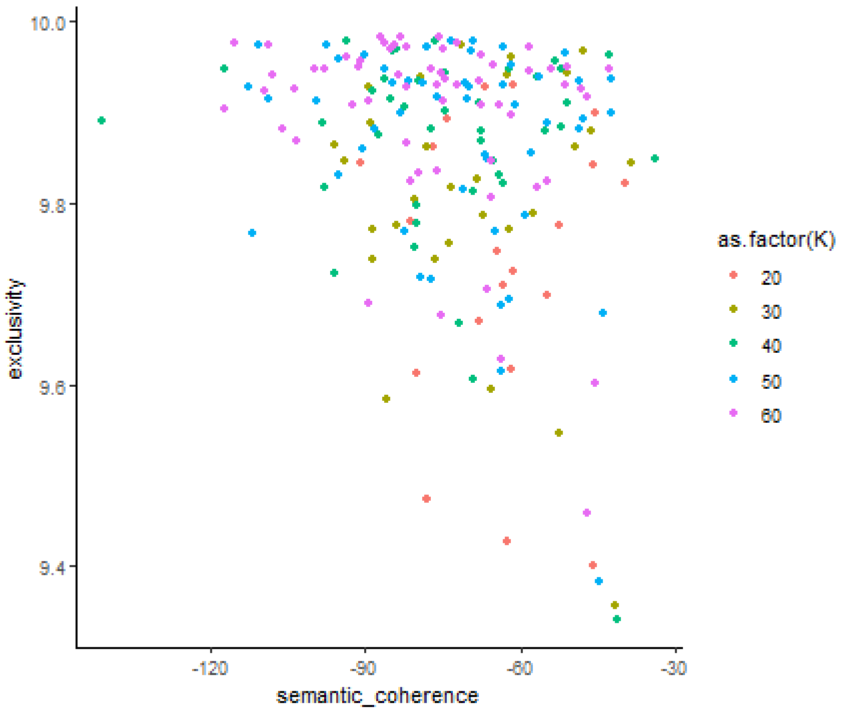
```
model_scores <- guardian_models %>%
  mutate(exclusivity = map(topic_model, exclusivity),
         semantic_coherence = map(topic_model, semanticCoherence, stm_dfm$documents)) %>%
  select(K, exclusivity, semantic_coherence)

model_scores
```

```
## # A tibble: 5 x 3
##       K exclusivity semantic_coherence
##   <dbl> <list>      <list>
## 1    20 <dbl [20]>  <dbl [20]>
## 2    30 <dbl [30]>  <dbl [30]>
## 3    40 <dbl [40]>  <dbl [40]>
## 4    50 <dbl [50]>  <dbl [50]>
## 5    60 <dbl [60]>  <dbl [60]>
```

Model selection

…and plot the values for all models:

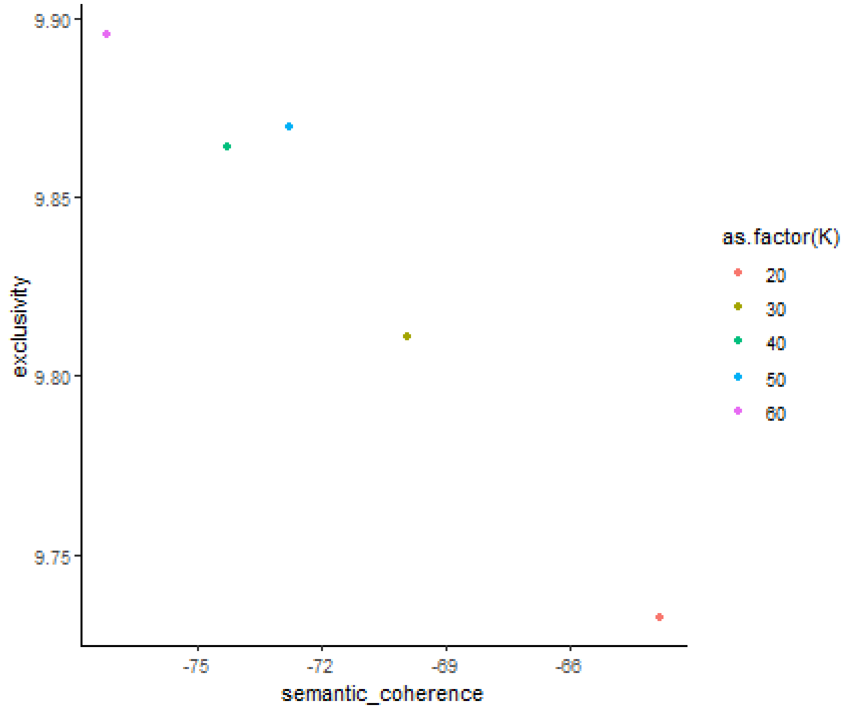
```
model_scores %>%
  unnest(c(exclusivity, semantic_coherence))
  ggplot(aes(x = semantic_coherence, y = excl
  geom_point() +
  theme_classic()
```



Model selection

To more easily compare models, we let's summarize both measures per model. This neatly shows the common trade-off, but it seems like the 40-topic solution may be a good start:

```
model_scores %>%
  unnest(c(exclusivity, semantic_coherence))
  group_by(K) %>%
  summarize(exclusivity = mean(exclusivity),
            semantic_coherence = mean(semanti
  ggplot(aes(x = semantic_coherence, y = excl
  geom_point() +
  theme_classic()
```



Model intepretation

Now to the fun stuff: What actually *are* our topics? First, let's extract our (for now) final model from the many models we calculated:

```
guardian_stm_40 <- guardian_models %>%
  filter(K == 40) %>%
  pull(topic_model) %>%
  .[[1]]

guardian_stm_40
```

```
## A topic model with 40 topics, 9965 documents and a 5165 word dictionary.
```

Model intepretation

We can extract the most important words per topic with the `labelTopics()` function. Apart from the actual word probabilities per topic, this also includes three additional ways of finding important words. For example, FREX (*frequency-exclusivity*) is the ratio of word frequency and word exclusivity per topic.

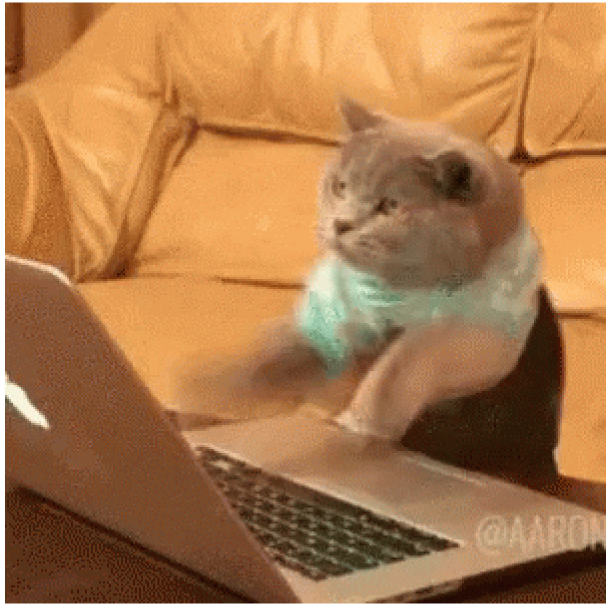
```
terms <- labelTopics(guardian_stm_40)
terms
```

```
## Topic 1 Top Words:
##       Highest Prob: local, city, new, london, people, council, building
##       FREX: local, housing, building, cities, city, town, streets
##       Lift: bridge, buildings, towns, housing, bike, traffic, building
##       Score: bridge, city, housing, local, residents, council, town
## Topic 2 Top Words:
##       Highest Prob: travel, de, french, france, two, german, flight
##       FREX: passengers, flight, flights, ship, airport, travel, crew
##       Lift: en, passenger, passengers, tourists, airport, railway, greece
##       Score: en, passengers, flights, france, french, travel, de
## Topic 3 Top Words:
##       Highest Prob: year, number, people, data, last, since, uk
##       FREX: figures, average, compared, increase, higher, rate, rise
##       Lift: grey, statistics, average, compared, figures, risen, proportion
##       Score: grey, data, average, figures, increase, rate, rates
```

Model intepretation

Exercise 1: Topic model interpretation

Try to label the topics from this model. Are there any topics that are problematic or stick out otherwise?



Model intepretation

To extract the actual probability values, let's make use of the good ol' `tidy()` function again. If applied to an STM object, this by default extracts the $V \times K$ matrix (called β in STM):

```
terms_probs <- tidy(guardian_stm_40, matrix = "beta")
terms_probs
```

```
## # A tibble: 206,600 x 3
##    topic term       beta
##    <int> <chr>     <dbl>
##  1     1 100m  4.82e-14
##  2     2 100m  5.57e-24
##  3     3 100m  4.17e-16
##  4     4 100m  7.52e- 5
##  5     5 100m  1.90e- 5
##  6     6 100m  2.03e- 7
##  7     7 100m  9.58e-13
##  8     8 100m  4.18e-30
##  9     9 100m  2.26e- 7
## 10    10 100m  8.11e- 9
## # ... with 206,590 more rows
```

Model interpretation

All beta values add up to 1 per topic:

```
terms_probs %>%
  group_by(topic) %>%
  summarise(sum_beta = sum(beta))
```

```
## # A tibble: 40 x 2
##    topic sum_beta
##    <int>    <dbl>
##  1     1        1
##  2     2        1
##  3     3        1
##  4     4        1
##  5     5        1
##  6     6        1
##  7     7        1
##  8     8        1
##  9     9        1
## 10    10        1
## # ... with 30 more rows
```

Model interpretation

To extract the $D \times K$ matrix (called γ in STM), simply pass `matrix = "gamma"` to `tidy()`:

```
doc_probs <- tidy(guardian_stm_40, matrix = "gamma", document_names = stm_dfm$meta$title)
doc_probs
```

```
## # A tibble: 398,600 x 3
##    document                                                     topic   gamma
##    <chr>                                                        <int>   <dbl>
##  1 We know this disaster is unprecedented – no amount of Scott Mo~     1 0.00218
##  2 Mariah Carey's Twitter account hacked on New Year's Eve             1 0.00319
##  3 Australia weather forecast: dangerous bushfire and heatwave co~     1 0.00610
##  4 TV tonight: Sherlock's writers get their teeth into Dracula        1 0.00417
##  5 Shipping fuel regulation to cut sulphur levels comes into force     1 0.00444
##  6 Western Balkans left 'betrayed' by EU over membership talks        1 0.00134
##  7 Welcome to the roaring 2020s – inside the 3 January edition of~     1 0.00557
##  8 The Power of Bad and How to Overcome It review – professional ~     1 0.00594
##  9 Top 10 books about new beginnings                                   1 0.00220
## 10 Three cities, VAR and a $15m prize – ATP Cup prepares for laun~     1 0.00764
## # ... with 398,590 more rows
```

Model interpretation

Gamma values add up to 1 per document:

```
doc_probs %>%
  group_by(document) %>%
  summarise(sum_gamma = sum(gamma))
```

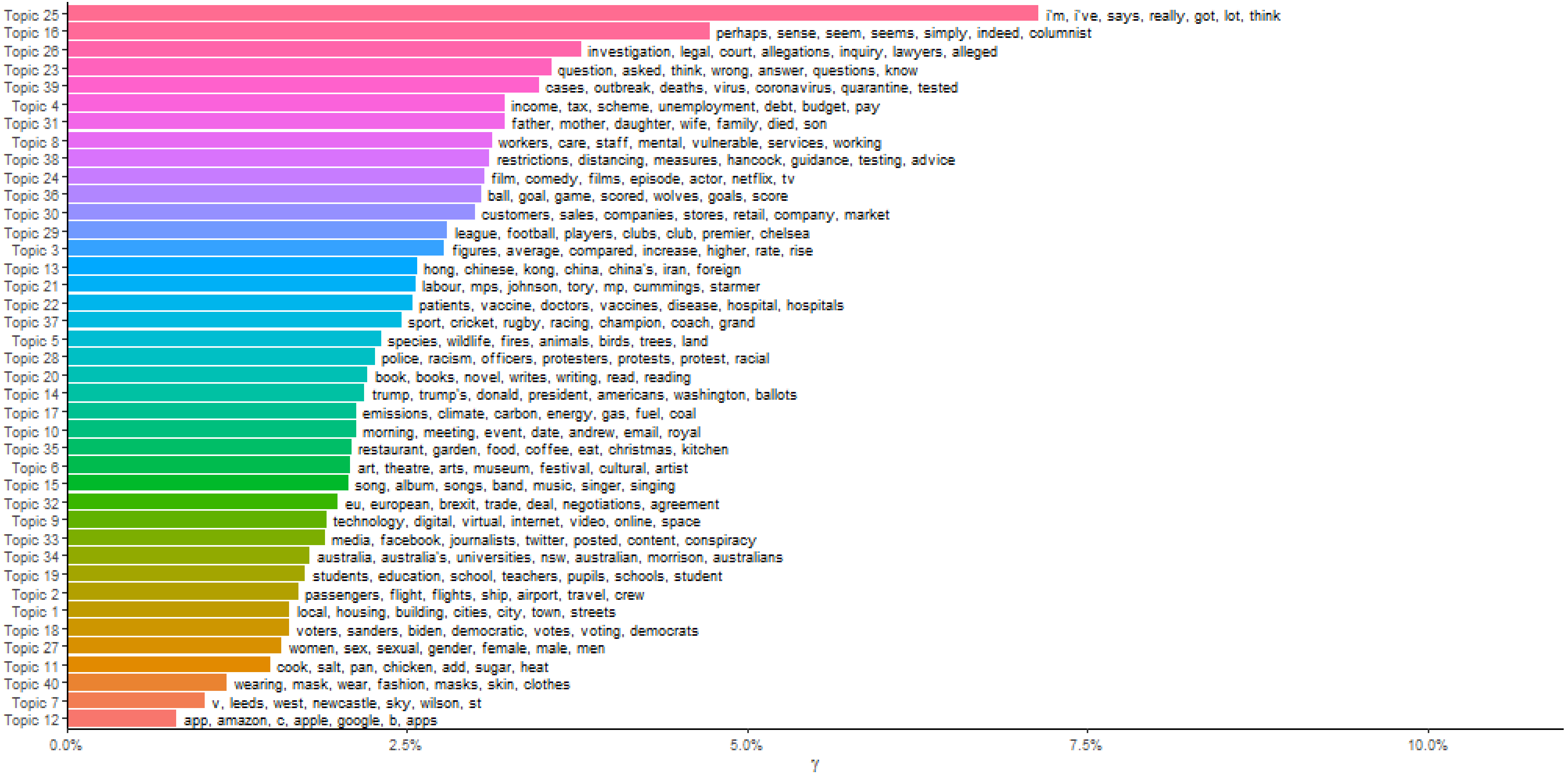
```
## # A tibble: 9,881 x 2
##    document                                                           sum_gamma
##    <chr>                                                                  <dbl>
##  1 '$1,000 per person should be the baseline': Andrew Yang on direct ~         1
##  2 'A beautiful change': Australia in bloom after drought-breaking ra~         1
##  3 'A chance to be more than a number': the female inmates podcasting~         1
##  4 'A climate change-scale problem': how the internet is destroying us         1
##  5 'A cry for help': Fifth of New Zealand high school pupils exposed ~         1
##  6 'A defining moment in the Middle East': the killing of Qassem Sule~         1
##  7 'A different twist': how school nativity plays have adapted to the~         1
##  8 'A game changer'. The UK's first LGBTQ+ extra-care housing scheme ~         1
##  9 'A ghost-town, tumbleweed quality': New York shuts down over coron~         1
## 10 'A giant has fallen': anti-apartheid activist Denis Goldberg dies ~         1
## # ... with 9,871 more rows
```

Model interpretation

One common way of reporting topic models is by plotting topic proportions and most important words together:

```
top_terms <- tibble(topic = terms$topicnums,
                    frex = apply(terms$frex, 1, paste, collapse = ", "))

gamma_by_topic <- doc_probs %>%
  group_by(topic) %>%
  summarise(gamma = mean(gamma)) %>%
  arrange(desc(gamma)) %>%
  left_join(top_terms, by = "topic") %>%
  mutate(topic = paste0("Topic ", topic),
         topic = reorder(topic, gamma))

gamma_by_topic %>%
  ggplot(aes(topic, gamma, label = frex, fill = topic)) +
  geom_col(show.legend = FALSE) +
  geom_text(hjust = 0, nudge_y = 0.0005, size = 3) +
  coord_flip() +
  scale_y_continuous(expand = c(0, 0), limits = c(0, 0.11), labels = scales::percent) +
  theme_classic() +
  theme(panel.grid.minor = element_blank(), panel.grid.major = element_blank()) +
  labs(x = NULL, y = expression(gamma))
```

Model interpretation



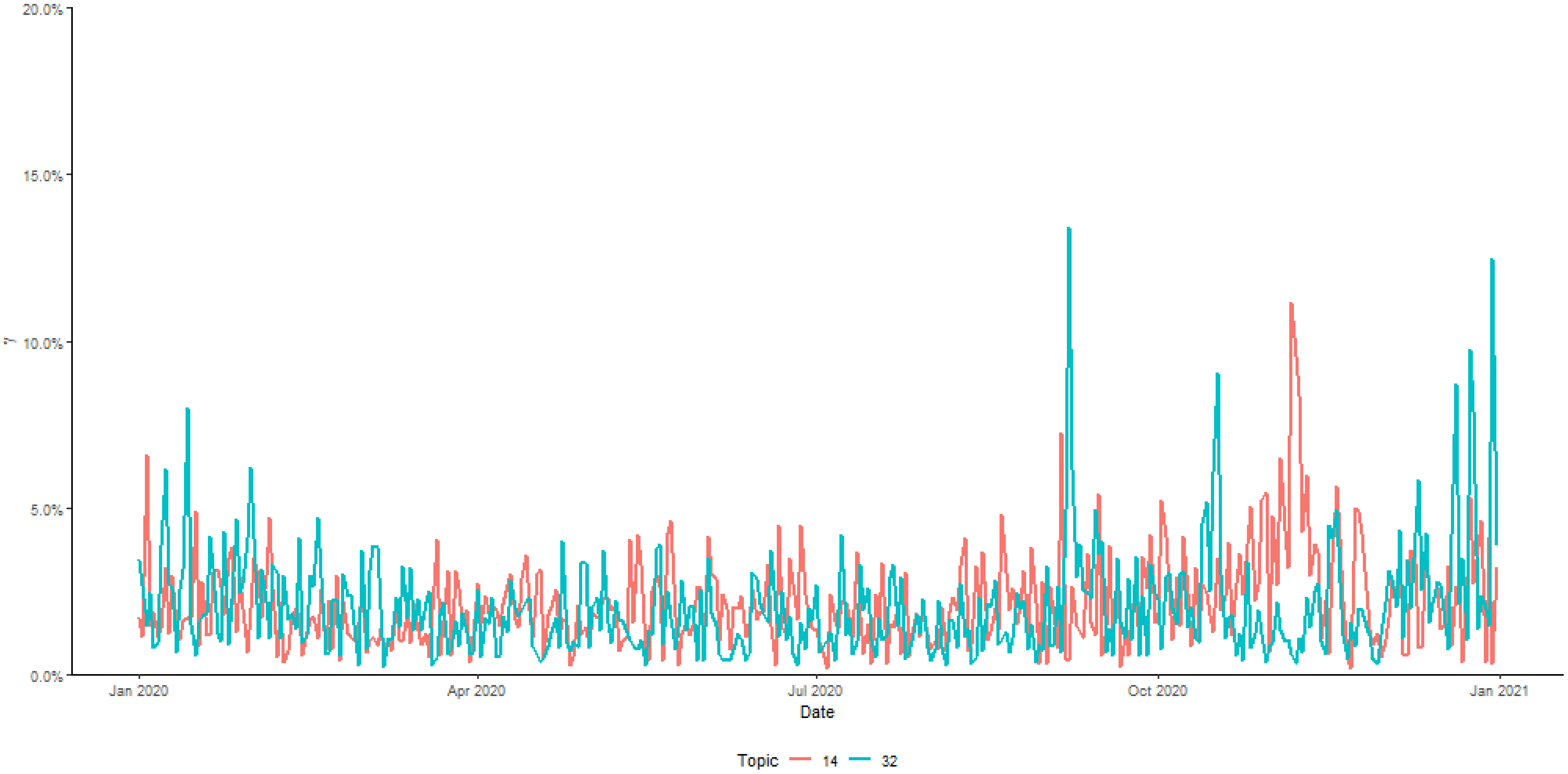
Model interpretation

Of course, we can now also make use of other document variables, for example, to show topic distribution over time. For example, let's compare topic 14 (US election terms) and 32 (Brexit terms):

```
doc_probs %>%
  left_join(guardian_tibble, by = c("document" = "title")) %>%
  mutate(day = lubridate::date(date)) %>%
  group_by(topic, day) %>%
  summarise(n = n(),
            gamma = mean(gamma),
            .groups = "drop") %>%
  mutate(topic = as_factor(topic)) %>%
  filter(topic %in% c(14, 32)) %>%
  ggplot(aes(x = day, y = gamma, color = topic, fill = topic)) +
  geom_line(size = 1) +
  theme_classic() +
  theme(panel.grid.minor = element_blank(),
        panel.grid.major.x = element_blank(),
        legend.position = "bottom") +
  scale_y_continuous(expand = c(0, 0), limits = c(0, 0.2), labels = scales::percent) +
  labs(x = "Date", y = expression(gamma), color = "Topic", fill = "Topic")
```

Model interpretation

Universität
Konstanz



Adding covariates

Universität Konstanz

Apart from just comparing topics by document meta variables after modeling, we can also explicitly model relationships between topics and those variables by adding them as covariates that predict topic prevalance in the model:

```
guardian_stm_40_cov <- stm(documents = stm_dfm$documents,
                           vocab = stm_dfm$vocab,
                           prevalence = ~ stm_dfm$meta$pillar,
                           K = 40,
                           verbose = FALSE)
guardian_stm_40_cov
```

```
## A topic model with 40 topics, 9965 documents and a 5165 word dictionary.
```

Adding covariates

We can then extract the effects with `estimateEffect()` function:

```
stm_40_effects <- estimateEffect(1:40 ~ pillar, guardian_stm_40_cov, stm_dfm$meta)
```

This provides regression tables per topic for the covariate effects:

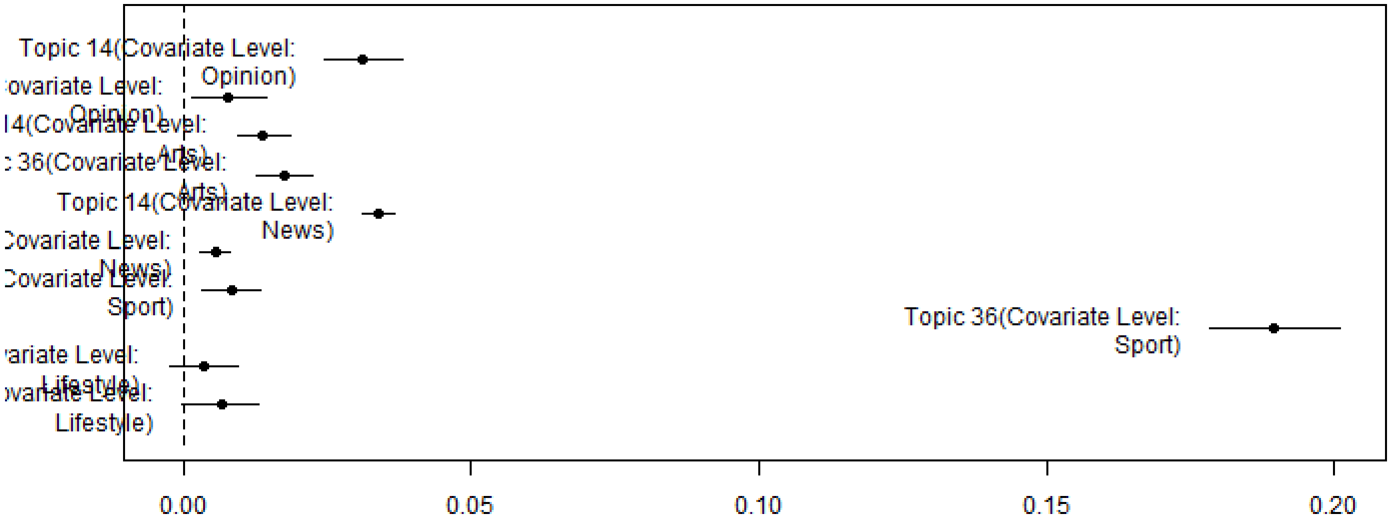
```
summary(stm_40_effects, topics = c(14))
```

```
##
## Call:
## estimateEffect(formula = 1:40 ~ pillar, stmobj = guardian_stm_40_cov,
##     metadata = stm_dfm$meta)
##
##
## Topic 14:
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.013826   0.002346   5.893 3.91e-09 ***
## pillarLifestyle -0.010491   0.003745  -2.801   0.0051 **
## pillarNews      0.019974   0.002773   7.204 6.27e-13 ***
```

Adding covariates

STM effects objects also have a `plot()` function:

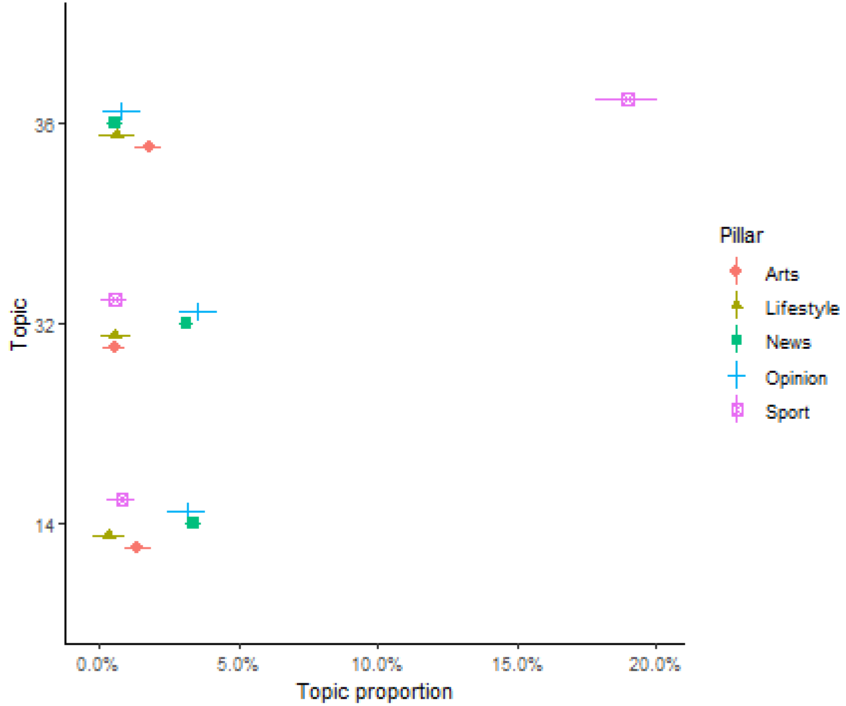
```r
plot(stm_40_effects, covariate = "pillar", topics = c(14, 36))
```



Adding covariates

Use the `stminsights` package to extract the values and have more options in plotting covariate effects:

```
stminsights::get_effects(stm_40_effects, "pil
  filter(topic %in% c(14, 32, 36)) %>%
  ggplot(aes(x = topic, y = proportion, ymin
  geom_pointrange(position = position_dodge(.
  coord_flip() +
  theme_classic() +
  scale_y_continuous("Topic proportion", labe
  labs(x = "Topic", color = "Pillar", shape =
```



Keyword-assisted topic models

Keyword-assisted topic models with keyATM

Universität Konstanz

A more recent expansion of topic models are called **keyword-assisted topic models**. These models somewhat combine deductive and inductive approaches, by mainly following the unsupervised topic modeling procedure, but allow the specification of a-priori topics with keywords beforehand.

In R, the `keyATM` package may be used to fit keyword-assisted topic models:

```
install.packages("keyATM")
library(keyATM)
```

```
## keyATM 0.4.0 successfully loaded.
##  Papers, examples, resources, and other materials are at
##  https://keyatm.github.io/keyATM/
```

Keyword-assisted topic models with keyATM

Again, keyATM uses it's own format for modeling, but also provides a conversion function:

```
keyATM_docs <- keyATM_read(texts = trimmed_dfm)
```

```
## Using quanteda dfm.
```

Defining a-priori topics with keywords

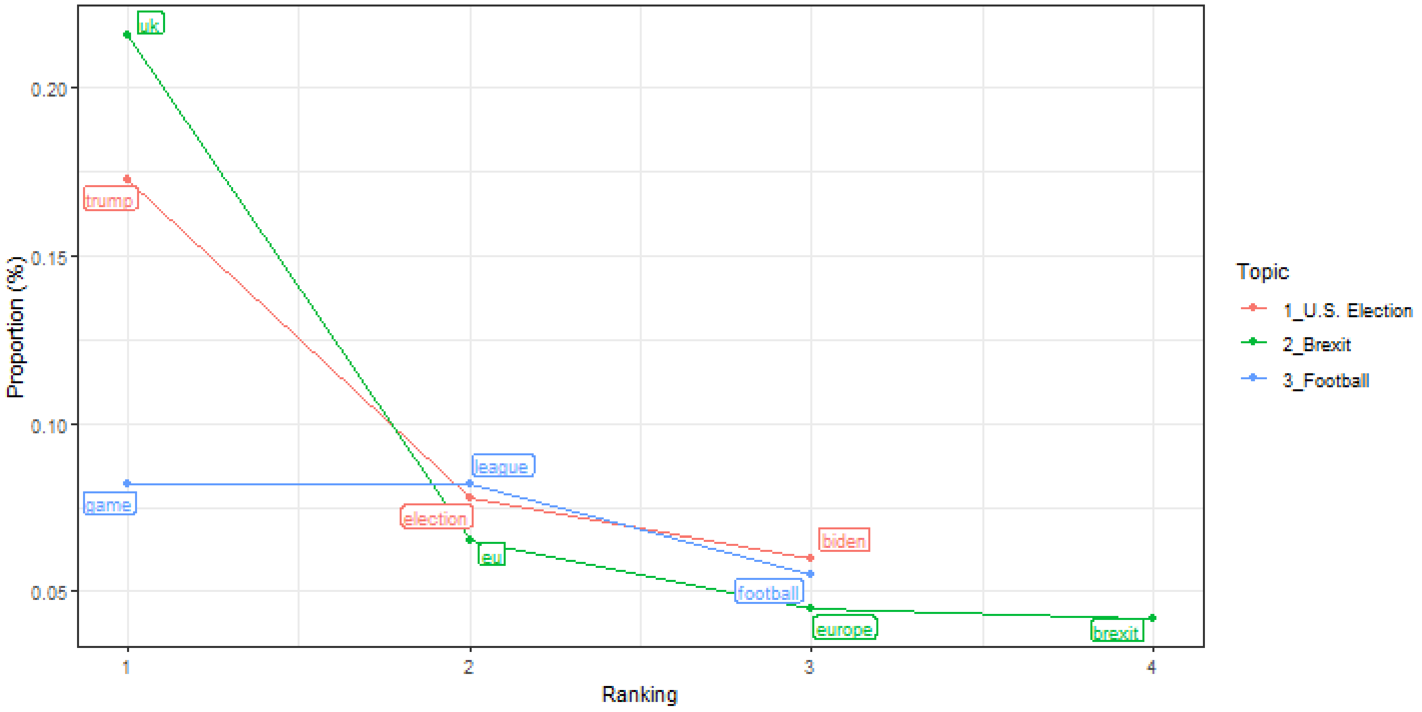
Let's work with the guardian corpus again, but this time, add some a-priori topics to the modell. We first create a named list of a-priori topics and associated keywords:

```r
keywords <- list(
  "U.S. Election" = c("biden", "trump", "election"),
  "Brexit" = c("brexit", "uk", "europe", "eu"),
  "Football" = c("football", "league", "game")
)
```

Defining a-priori topics with keywords

`keyATM` kindly provides a function `visualize_keywords()` to inspect whether our keywords are actually useful by plotting their relative frequency. The authors suggest a proportion of at least 0.1% per keyword, but for larger corpora and more distinctive topics, lower numbers may be okay as well:

```
visualize_keywords(keyATM_docs, keywords)
```



Model fitting

Universität Konstanz

We fit the model using the `keyATM()` function using the following arguments:

- `docs` defines our DFM, which we have converted to the `keyATM` format.
- `keywords` defines our a-priori topics with associated keywords.
- `no_keyword_topics` defines the number of additional topics the model should estimate.
- `model` specifies the model type; we are going to use the simple `"base"` model, but note that you may also use additional models that, for example, allow for covariate specification. See the offical documentation for more details.

```
guardian_keyatm <- keyATM(docs = keyATM_docs,
                          keywords = keywords,
                          no_keyword_topics = 37,
                          model = "base")
```

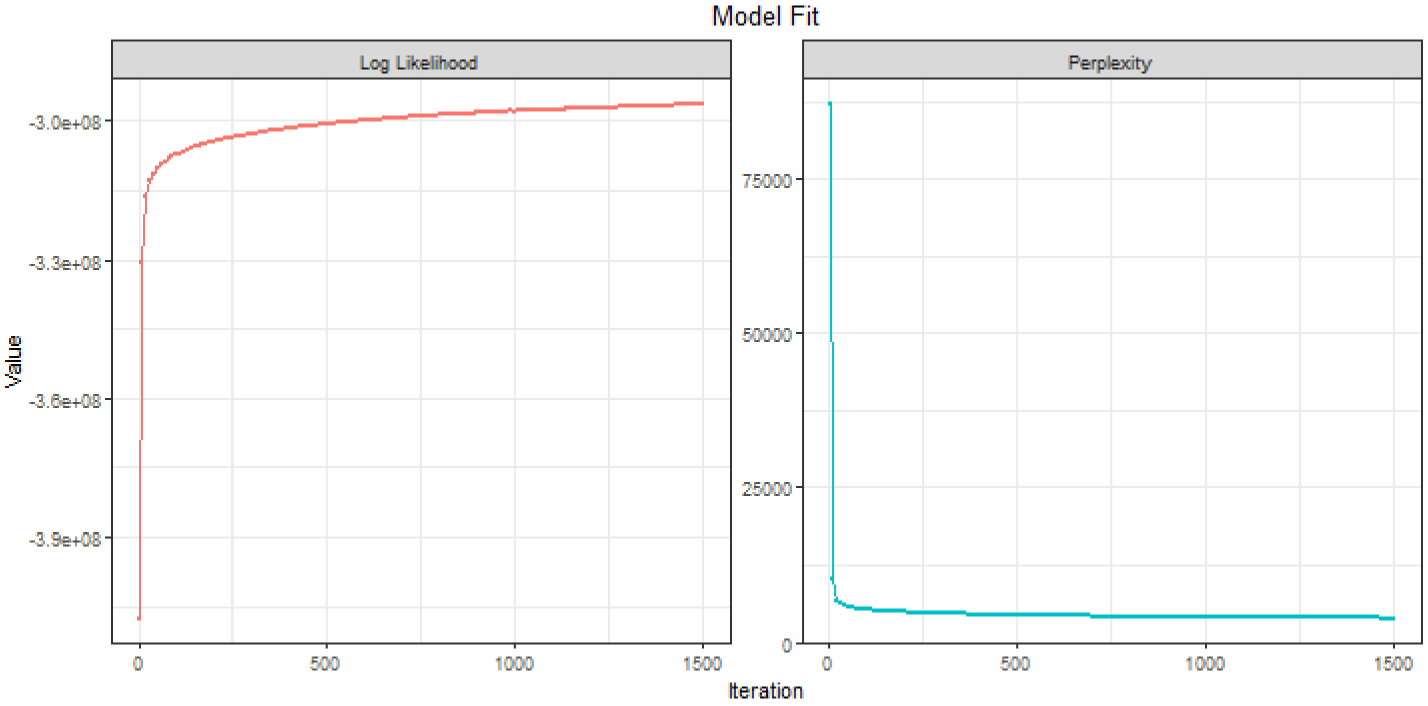
```
guardian_keyatm <- readRDS("offline_data/5/guardian_keyatm.rds")
```

Model selection

Universität Konstanz

We can measure and compare model fit using the `plot_modelfit()` function, which plots two model fit measures against the model fit iterations. *Log-likelihood* should stabilize on a high value, *perplexity* on a low value over time to indicate good model fit:

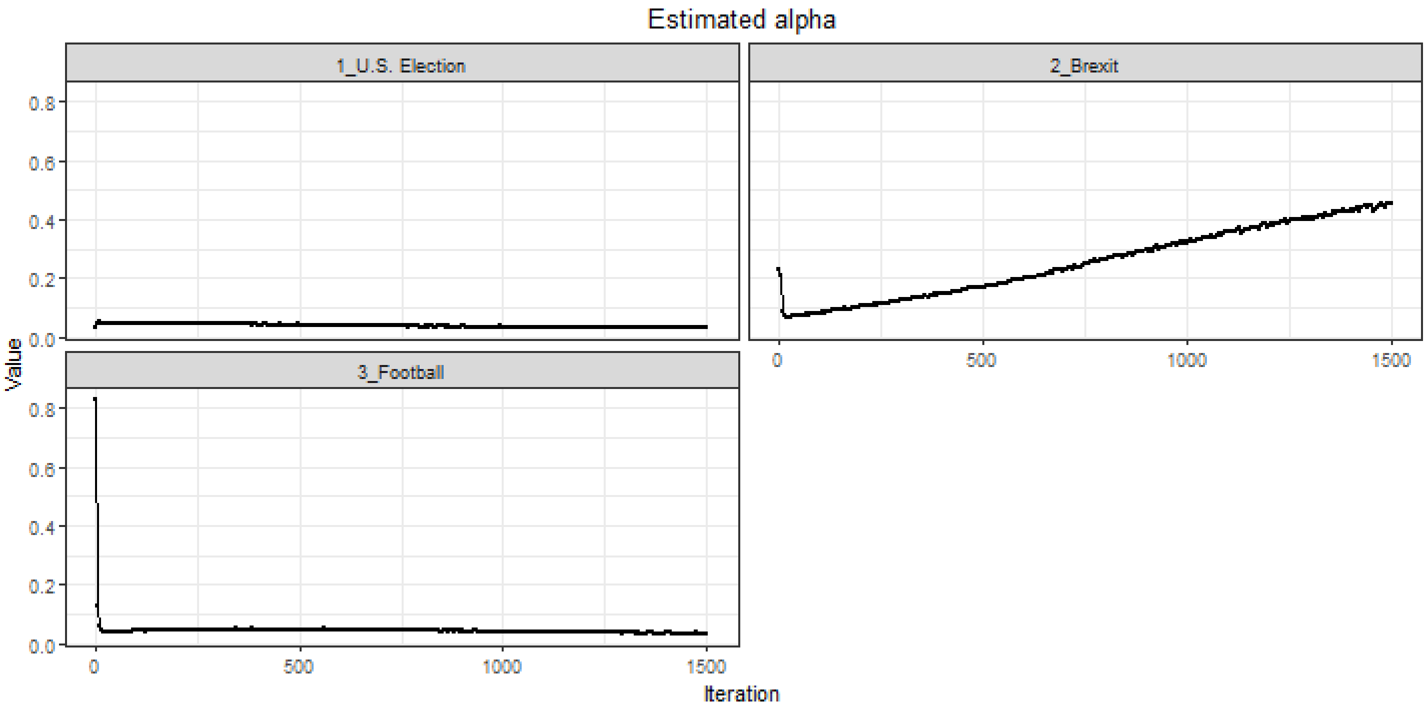
```
plot_modelfit(guardian_keyatm)
```



Model selection

We can also plot α, the document-topic distribution prior, against model iterations. Again, values should stabilize over time to indicate good model fit. This indicates that the Brexit topic is probably not well defined by the keywords we chose:

```
plot_alpha(guardian_keyatm)
```



Model interpretation

Similarly to before, `top_words()` reports the most important words per topic. Note that the checkmark symbol indicates a-priori keywords for topics, numbers in square brackets [ ] indicate keywords from a a-priori topics appearing in other topics:

```
top_words(guardian_keyatm, n = 7)
```

```
##       1_U.S. Election      2_Brexit        3_Football Other_1 Other_2
## 1    trump [<U+2713>] uk [<U+2713>] league [<U+2713>]     art climate
## 2 election [<U+2713>]    government   game [<U+2713>]  london  energy
##         Other_3 Other_4    Other_5 Other_6 Other_7 Other_8 Other_9 Other_10
## 1        report vaccine  australia    year  family    like    film   people
## 2 investigation  health australian company    died  people    show   health
##   Other_11 Other_12 Other_13 Other_14   Other_15 Other_16    Other_17 Other_18
## 1  workers students     fire   people government  climate coronavirus    local
## 2     work   school    water     says    economy  species       cases     city
##   Other_19 Other_20 Other_21 Other_22 Other_23  Other_24 Other_25   Other_26
## 1     food     care business     book  fashion    people    china    johnson
## 2      add   health lockdown    world    masks political  chinese government
##   Other_27 Other_28 Other_29 Other_30 Other_31 Other_32 Other_33 Other_34
## 1    women   police    music    first    media    court     says     like
## 2    black   people    album    years     news     case      i'm     time
##     Other_35 Other_36 Other_37
```

Model interpretation

Comparably, `top_docs()` reports the most important documents (index of the model DFM) per topic:

```
top_docs(guardian_keyatm, n = 1)
```

```
##                   apply(x$theta, 2, measuref)
## 1_U.S. Election                          8556
## 2_Brexit                                 6995
## 3_Football                               5960
## Other_1                                  4231
## Other_2                                  7919
## Other_3                                  5835
## Other_4                                  5742
## Other_5                                  1037
## Other_6                                   235
## Other_7                                  7708
## Other_8                                  1073
## Other_9                                   891
## Other_10                                 7638
## Other_11                                 4972
## Other_12                                 6892
## Other_13                                 7972
## Other_14                                 2020
```

Model interpretation

Sadly, `keyATM` objects are not yet compatible with `tidy()`. However, we can access the $V \times K$ (called ϕ in this case) and $D \times K$ (called θ in this case) matrices directly from the model object:

```
guardian_keyatm$phi
guardian_keyatm$theta
```

Model intepretation

Let's transform ϕ and extract the 7 most important words per topic:

```
top_terms <- guardian_keyatm$phi %>%
  t() %>%
  as_tibble(rownames = "word") %>%
  pivot_longer(-word, names_to = "topic", values_to = "phi") %>%
  group_by(topic) %>%
  top_n(7, phi) %>%
  arrange(topic, desc(phi)) %>%
  group_by(topic) %>%
  summarise(top_words = paste(word, collapse = ", "), .groups = "drop")

top_terms
```

```
## # A tibble: 40 x 2
##    topic          top_words
##    <chr>          <chr>
##  1 1_U.S. Electi~ trump, election, president, biden, us, trump's, donald
##  2 2_Brexit       uk, government, could, new, last, time, make
##  3 3_Football     league, game, players, season, team, football, last
##  4 Other_1        art, london, arts, theatre, work, artists, festival
##  5 Other_10       people, health, coronavirus, home, covid-19, government, publ~
```

Model intepretation

Similarly, extract mean topic proportions from θ. Again, the proportion of the Brexit topic indicates that this was probably not the best specified topic:

```
top_topics <- guardian_keyatm$theta %>%
  as_tibble(rownames = "document") %>%
  pivot_longer(-document, names_to = "topic", values_to = "theta") %>%
  group_by(topic) %>%
  summarise(mean_theta = mean(theta), .groups = "drop") %>%
  arrange(desc(mean_theta))

top_topics
```

```
## # A tibble: 40 x 2
##    topic          mean_theta
##    <chr>               <dbl>
##  1 2_Brexit           0.239
##  2 3_Football         0.0765
##  3 Other_34           0.0637
##  4 Other_30           0.0514
##  5 Other_14           0.0439
##  6 Other_8            0.0366
##  7 Other_22           0.0350
```

Model intepretation

To create a similar plot as before, we can join both tibbles:

```
top_topics %>%
  left_join(top_terms, by = "topic") %>%
  mutate(topic = reorder(topic, mean_theta)) %>%
  ggplot(aes(topic, mean_theta, label = top_words, fill = topic)) +
  geom_col(show.legend = FALSE) +
  geom_text(hjust = 0, nudge_y = 0.0005, size = 3) +
  coord_flip() +
  scale_y_continuous(expand = c(0, 0), limits = c(0, 0.4), labels = scales::percent) +
  theme_bw() +
  theme(panel.grid.minor = element_blank(),
        panel.grid.major = element_blank()) +
  labs(x = NULL, y = expression(theta))
```

Model interpretation

Validating topic models

Validating topic models

As topic models will *always* output the desired number of topics, again, validation is key. For topic models, the following validation steps are common:

- Computing data fit indices (e.g., semantic coherence, exclusivity)
- Manually labelling and intepreting topics (duh)
- Investigating meaningful relationships of results with other variables in the data (e.g., a terrorism topic should lead to higher scores in the aftermath of terrorist attacks)

Furthermore, for manual validation, we usually are not able to provide gold standards, as we did not define the topics ourselves. However, two methods were developed to manually validate how good topics can be interpreted by humans:

- **Word intrusion test**: Randomly draw n words with high probabilities and 1 word with low probability from the same topic distribution. Human coders should then be able to identify the *intruder* word.
- **Topic intrusion test**: Randomly drawn n topics with high probabilities and 1 topic with low probability from the same document distribution. Human coders should then be able to identify the *intruder* topic after reading through the document.
- In both cases, we can then compute the precision of repeated word/topic intrusion tests for multiple topics/documents.

Validating topic models with `oolong`

Both tests are implemented in the `oolong` package know from last time:

```
library(oolong)
```

The workflow is quite simple:

- Use `wi()` (word intrusion), `wsi()` (word-set intrusion; variant of word intrusion with sets of words instead of single words), and `ti()` to create the test object with the model object as input.
- Use the associated method to do the actual test (`$do_xxx()`).
- `$lock()` the object to display results.
- `oolong()` objects can be cloned before doing the test to accomodate for multiple coders with `clone_oolong()`.
- Use `summarize_oolong()` to compare the results of multiple tests.

Word intrusion tests

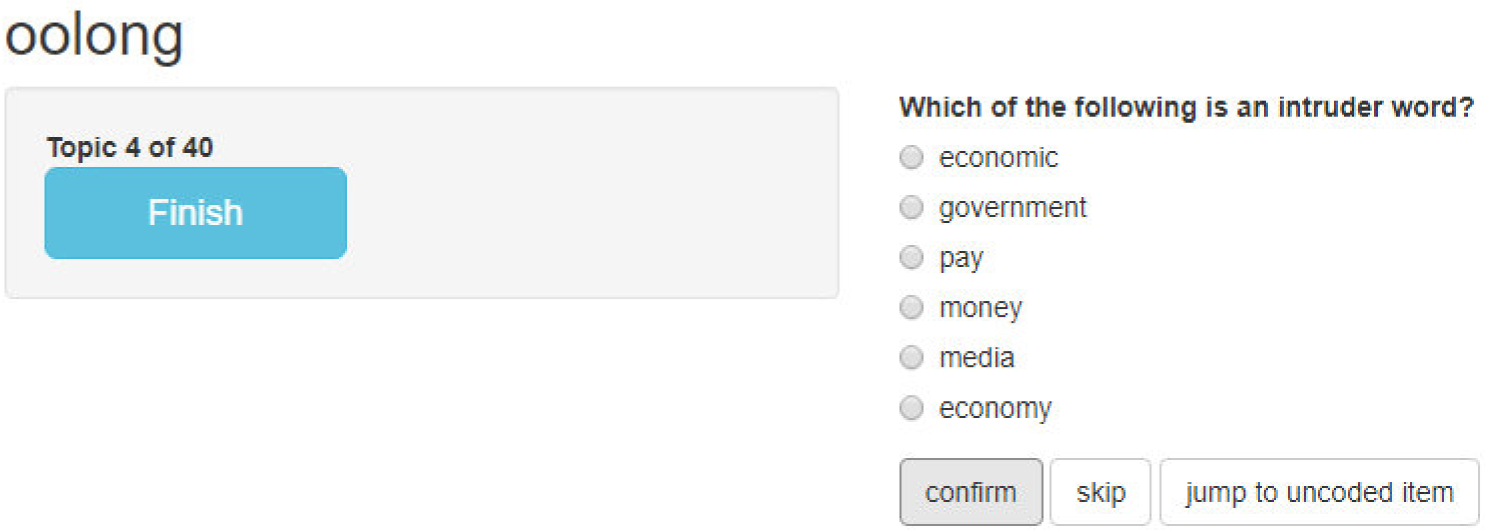
Universität Konstanz

Example: Word intrusion test

```r
# Create and clone objects
wi_test_coder_1 <- wi(guardian_stm_40_cov, userid = "Coder 1")
wi_test_coder_2 <- clone_oolong(wi_test_coder_1, userid = "Coder 2")

# Do the test
wi_test_coder_1$do_word_intrusion_test()
wi_test_coder_2$do_word_intrusion_test()

# Lock
wi_test_coder_1$lock()
wi_test_coder_2$lock()

# Summarize
summarize_oolong(wi_test_coder_1, wi_test_coder_2)
```

Word intrusion tests

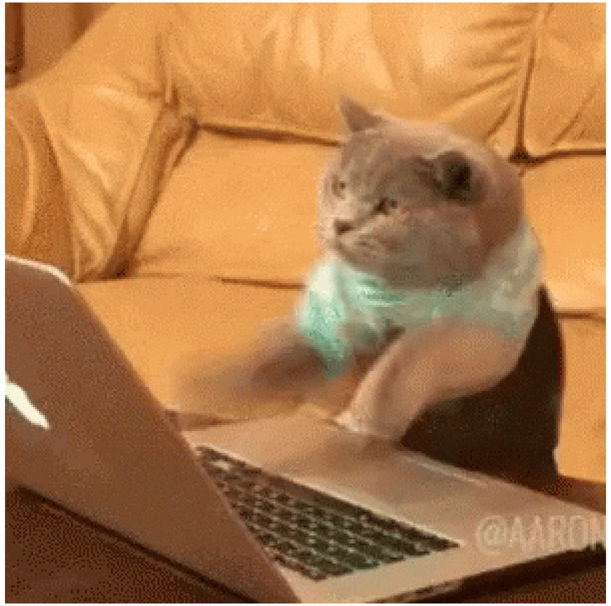
Example: Word intrusion test



Word-set intrusion tests

Exercise 2: Validating topic models

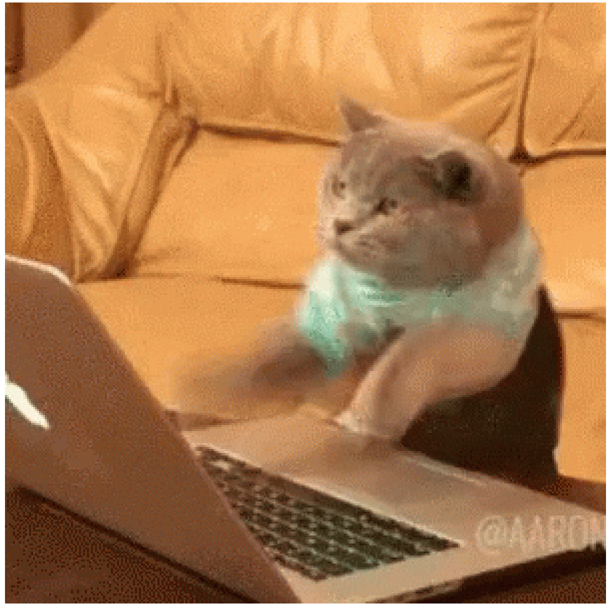
Create an oolong object with `wsi()` on the `guardian_stm_40_cov` model and perform a word-set intrusion test. Good luck!



Topic intrusion tests

Exercise 3: Validating topic models

Create an oolong object with `ti()` on the `guardian_stm_40_cov` model and perform a topic intrusion test. Good luck!



Thanks

Credits:

- Slides created with `xaringan`
- Title image by Marjan Blahn / Unsplash
- Coding cat gif by Memecandy/Giphy