Implementing STM in R

Promothesh Chatterjee[*]

[*]Copyright© by Promothesh Chatterjee. All rights reserved. No part of this note may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author.

Structural Topic Models (STMs) are a type of statistical model used to analyze text data. Imagine you have a large number of documents—like a library full of books—and you want to understand what they're about, but you don't have time to read each one individually. You could use a Structural Topic Model to help with this.

Here's how it works in simpler terms:

1. **Topics**: The STM assumes that each document is a mixture of different topics. A topic is a collection of words that frequently occur together. For example, in a document about cooking, words like "recipe", "ingredient", "cook", "bake", "stir" might frequently appear together, and these would form a "cooking" topic.
2. **Structural part**: The STM allows us to include additional information about the documents or the context in which they were written. For example, if we know the author of each document, or the date it was written, or the location it was written about, we can incorporate this information into the model. This helps us understand how the topics vary based on these factors. For example, we might find that certain topics are more common in documents written by a certain author, or at a certain time, or about a certain location. This is analogous to adding covariates in a regression framework.
3. **Analysis**: Once we've fit the model, we can analyze the topics that it found. For example, we can look at the most common words in each topic, or how the prevalence of each topic varies based on the additional information we included (the author, date, location, etc.). This gives us insights into the main themes in our collection of documents and how these themes relate to the context in which the documents were written.

In summary, a Structural Topic Model is a tool that helps us understand the main themes in a collection of documents and how these themes vary based on additional information we have about the documents.

```r
# Load necessary libraries
library(quanteda)
library(stm)

# Create synthetic dataset
reviews <- c("Great product, very high quality.",
             "Terrible customer service.",
             "Good value for money.",
             "Product broke after just a few days.",
             "Excellent product, and excellent customer service.",
             "Not worth the price.",
             "I love this product!",
             "The product didn't meet my expectations.",
             "Fantastic! Highly recommend.",
             "The customer service was very disappointing.")

ratings <- c(5, 1, 4, 1, 5, 2, 5, 2, 5, 1)
```

```r
regions <- c("North", "South", "East", "West", "North", "South", "East", "West", "North", "South")

purchase_location <- c("Online", "In-store", "Online", "In-store", "Online", "In-store", "Online", "In-s

user_type <- c("New", "Returning", "New", "Returning", "New", "Returning", "New", "Returning", "New", "R

# Combine into a data frame
data <- data.frame(reviews, ratings, regions, purchase_location, user_type)

# Prepare covariates
ratings <- as.numeric(data$ratings)
regions <- as.factor(data$regions)
purchase_location <- as.factor(data$purchase_location)
user_type <- as.factor(data$user_type)

# Process the text for STM
processed <- textProcessor(documents = data$reviews, metadata = data)
```

```
## Building corpus...
## Converting to Lower Case...
## Removing punctuation...
## Removing stopwords...
## Removing numbers...
## Stemming...
## Creating Output...
```

```r
out <- prepDocuments(processed$documents, processed$vocab, processed$meta)
```

```
## Removing 19 of 23 terms (19 of 32 tokens) due to frequency
## Removing 2 Documents with No Words
## Your corpus now has 8 documents, 4 terms and 13 tokens.
```

```r
# Define the number of topics
K <- 3

# Run STM
stm_model <- stm(documents = out$documents, vocab = out$vocab, K = K,
                 prevalence =~ ratings + regions + purchase_location + user_type,
                 data = out$meta, init.type="Spectral", verbose=FALSE)

# Display the topics
print(stm_model)
```

```
## A topic model with 3 topics, 8 documents and a 4 word dictionary.
```

```r
# Display the most probable words for each topic
top_words <- labelTopics(stm_model, n = 3)
print(top_words)
```

```
## Topic 1 Top Words:
##       Highest Prob: high, product, custom
##       FREX: high, product, custom
##       Lift: high, product, custom
##       Score: high, product, custom
## Topic 2 Top Words:
##       Highest Prob: product, custom, high
##       FREX: custom, product, high
##       Lift: custom, product, high
##       Score: custom, product, high
## Topic 3 Top Words:
##       Highest Prob: servic, product, custom
##       FREX: servic, product, custom
##       Lift: servic, product, high
##       Score: servic, custom, high
```

```r
# Estimate the effect of the covariates on the topics
effect <- estimateEffect(1:K ~ ratings + regions + purchase_location + user_type,
                         stm_model, meta = out$meta)
#print(effect)
```

This code performs a Structural Topic Model (STM) analysis on a synthetic dataset of product reviews. The objective of an STM analysis is to understand the main topics in a collection of documents (in this case, reviews), and how these topics relate to document-level covariates (in this case, ratings, regions, purchase location, and user type).

Here's a high-level summary of what each part of the code does:

1. The code first processes the reviews to create a document-term matrix (a mathematical matrix that describes the frequency of terms that occur in the collection of documents) and a vocabulary list. This is done using the `textProcessor` and `prepDocuments` functions.

2. An STM model is then fitted to the processed documents, vocabulary, and covariates. The number of topics is set to `K` (which you must define). The `stm` function is used to fit the model. The model's objective is to learn the main topics in the reviews and how these topics are related to the covariates.

Topic Prevalence: It measures the extent to which each topic is represented in the corpus. For example, if we have a topic about "customer service," its prevalence would tell us how common this topic is across all the reviews.

Covariates: These are additional pieces of information or metadata about the documents that might influence topic prevalence. In the provided code, the covariates are ratings, regions, purchase_location, and user_type. The prevalence formula (prevalence =~ ratings + regions + purchase_location + user_type) tells the model to consider how these covariates affect the distribution of topics.

Example: If ratings is a covariate, the model will assess how the rating given in each review influences the likelihood of certain topics appearing. A higher rating might be associated with topics related to product satisfaction, while lower ratings might be linked to topics about customer complaints.

3. After fitting the model, the code prints out the model using the `print` function. This provides an overview of the model, including the number of topics, the number of documents, and some other information.

4. The code then prints out the most probable words for each topic using the `labelTopics` function. This helps you interpret the topics by showing you the words that are most associated with each topic. For example, if a topic has the words "customer", "service", and "disappointing" as its most probable words, you might interpret this topic as being about negative customer service experiences.

Metrics Explained:

Highest Prob (Highest Probability): Description: This metric lists the words that have the highest probability of appearing in the topic. These are the words most frequently associated with the topic. Interpretation: In Topic 1, the words "high," "product," and "custom" are the most probable words, indicating they frequently occur in documents that are categorized under this topic.

FREX (FRequency and EXclusivity): Description: FREX is a weighted measure that balances the frequency of a word in the topic with its exclusivity to that topic. It aims to highlight words that are both common in the topic and relatively unique to it. Interpretation: For Topic 1, "high," "product," and "custom" are the top FREX words, suggesting they are not only common but also somewhat unique to this topic compared to others.

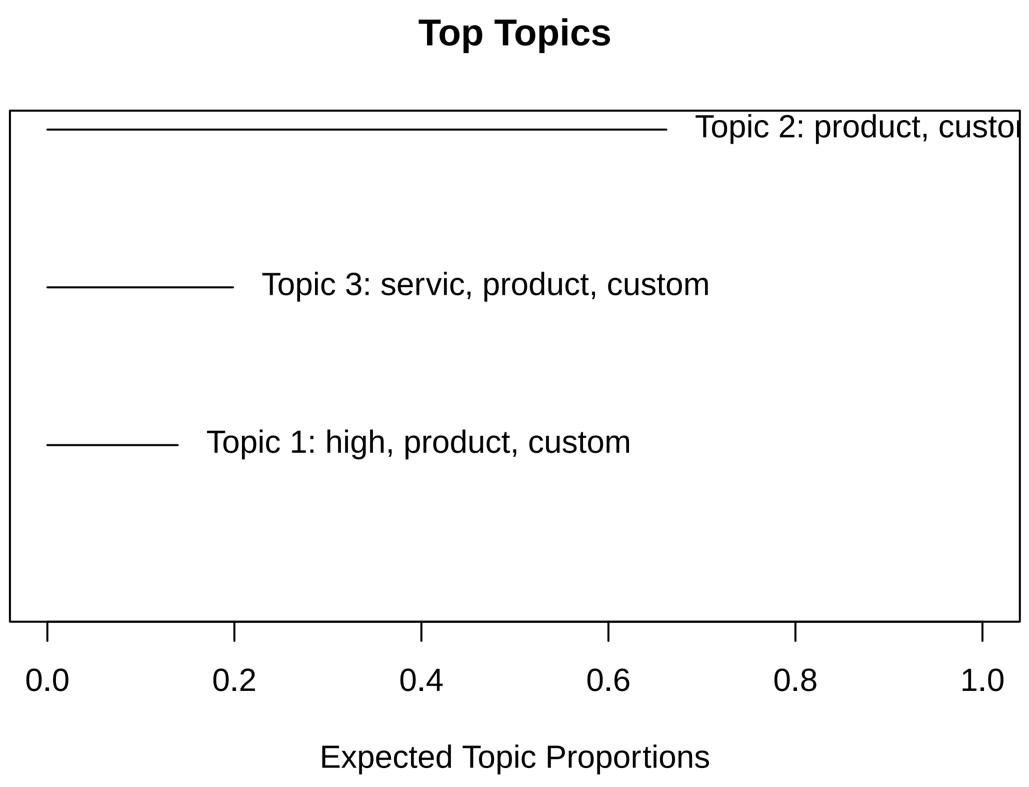
Lift: Description: Lift measures the importance of a word by considering its frequency in the topic relative to its overall frequency in the corpus. Higher lift values indicate words that are disproportionately frequent in the topic compared to their general usage. Interpretation: In Topic 1, the words "high," "product," and "custom" have high lift values, meaning they appear much more frequently in this topic than in the overall corpus.

Score: Description: Score is an internal measure used by the STM package to rank words within a topic. It often aligns closely with the highest probability metric. Interpretation: The top score words for Topic 1 are "high," "product," and "custom," reaffirming their significant association with this topic.

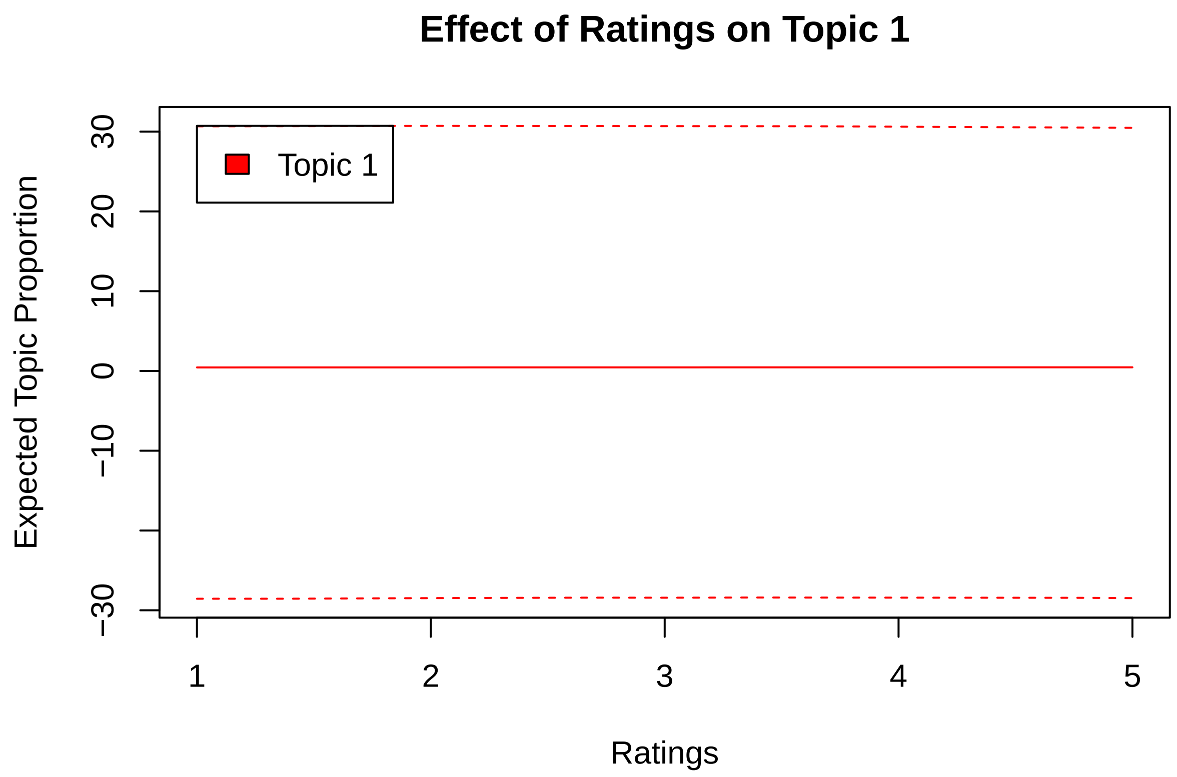
5. Then, the code estimates the effect of the covariates on the topics using the `estimateEffect` function. This can give you insight into how the topics relate to the covariates. For example, you might find

that negative topics are more prevalent in reviews with low ratings, or that certain topics are more prevalent in certain regions.

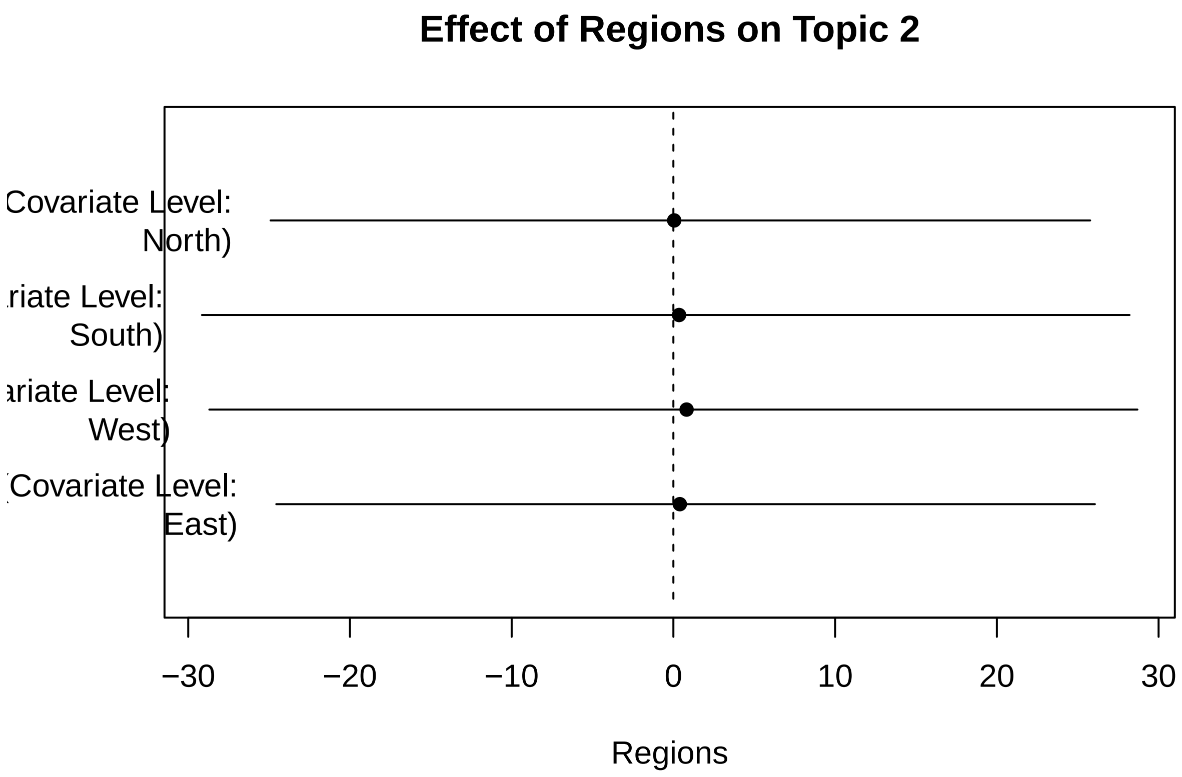
```r
plot(stm_model, type = "summary")
```



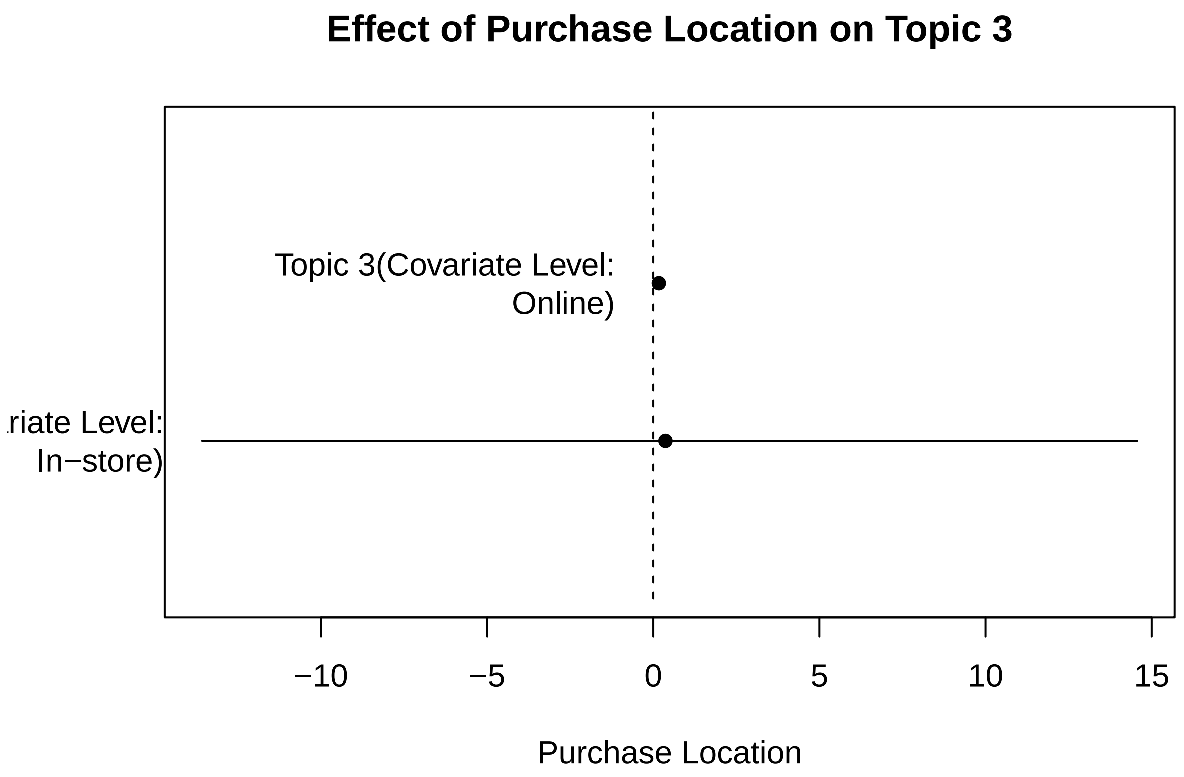
```r
plot(effect, covariate = "ratings", topics = 1,
     model = stm_model, method = "continuous", xlab = "Ratings", main = "Effect of Ratings on Topic 1")
```



```
plot(effect, covariate = "regions", topics = 2,
     model = stm_model, method = "pointestimate", xlab = "Regions", main = "Effect of Regions on Topic
```



```
plot(effect, covariate = "purchase_location", topics = 3,
     model = stm_model, method = "pointestimate", xlab = "Purchase Location", main = "Effect of Purchase
```



Components of the Summary Plot:

Topic Proportions:

The plot displays the proportion of each topic within the entire corpus. This shows how much of the text data is associated with each topic. Interpretation: A higher proportion indicates that the topic is more prevalent in the corpus. Top Words for Each Topic:

Alongside the proportions, the plot typically includes the most probable words for each topic. These words help to characterize and understand the theme of each topic. Interpretation: These words are the ones most strongly associated with each topic, giving insights into the main ideas or themes.

Topic Labels:

Each topic is labeled, usually with an identifier like "Topic 1," "Topic 2," etc. Interpretation: These labels help to differentiate between the various topics identified by the model.

6. Finally, we visualize the estimated effect of covariates on topic prevalence. - **ratings**: Continuous variable. - **regions**: Categorical variable. - **purchase_location**: Categorical variable.

Please note that the synthetic dataset used in this code is very simple, and the results might not be meaningful. In a real-world scenario, you would use a much larger and more complex dataset, and the interpretation of the results would depend on the specifics of the dataset and the STM model.

The Structural Topic Model (STM) package in R does not directly provide a method to compute perplexity or topic coherence. However, it provides a method called searchK that splits the data into a training set and a held-out set and fits STM models with different k to the training set. The held-out likelihood of each model on the held-out set is then computed, and the model with the highest held-out likelihood is selected as the best model.

Identifying K

- Here is how you might use searchK to identify a suitable k:

```r
# Load the libraries
library(stm)
library(quanteda)

# Create a synthetic dataset
texts <- c("I loved the product",
           "The service was terrible",
           "Great value for the price",
           "Product broke after a few uses",
           "I had a great experience",
           "Customer service was rude",
           "Good quality for the price",
           "Product was faulty")

# Additional factors
ratings <- c(5, 1, 4, 2, 5, 1, 4, 2)
regions <- c("North", "South", "East", "West", "North", "South", "East", "West")
purchase_location <- c("Online", "In store", "Online", "In store", "Online", "In store", "Online", "In s
user_type <- c("New", "Existing", "New", "Existing", "New", "Existing", "New", "Existing")

# Create a dataframe
df <- data.frame(texts, ratings, regions, purchase_location, user_type)

# Convert to corpus and preprocess
corpus <- corpus(df$texts)
docvars(corpus) <- df[, -1]  # assign document-level variables
tokens <- tokens(corpus, remove_punct = TRUE)
dfm <- dfm(tokens)

# Prepare the data for stm
out <- convert(dfm, to = "stm")

# Find k: Approach 2
```

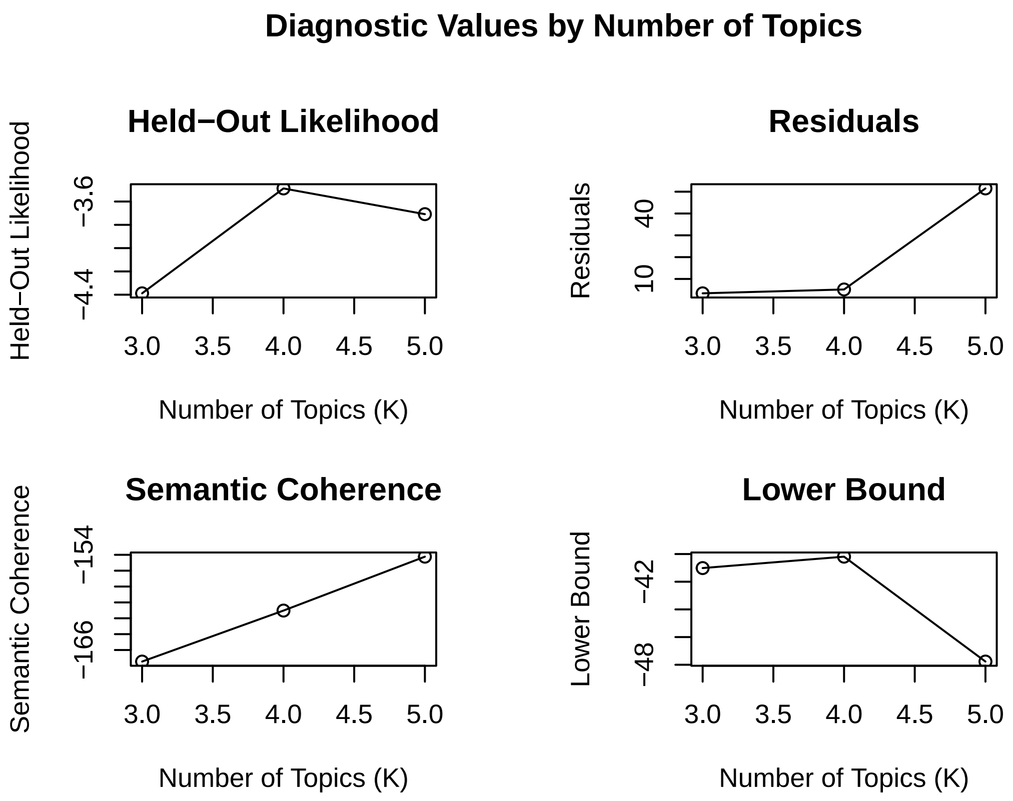
```r
set.seed(835)
system.time({
    findingk_ver2.searchK <- searchK(documents = out$documents,
                                    vocab = out$vocab,
                                    K = c(3,4,5), #specify K to try
                                    N = 7, # matching the size of the dataset
                                    proportion = 0.5, # default
                                    heldout.seed = 1234, # optional
                                    M = 10, # default
                                    cores = 1, # default
                                    prevalence =~ ratings + regions + purchase_location + user_type,
                                    max.em.its = 75,
                                    data = out$meta,
                                    init.type = "Spectral",
                                    verbose=FALSE)
})
```

```
##    user  system elapsed
##    0.64    0.04    0.71
```

```r
# Check the results
findingk_ver2.searchK
```

```
## $results
##   K    exclus    semcoh   heldout residual     bound    lbound em.its
## 1 3 6.762439 -167.4555 -4.388062 3.417498 -42.80575 -41.01399     75
## 2 4 6.765446 -161.0275 -3.487428 5.170899 -43.36869 -40.19064     75
## 3 5 6.779072 -154.2354 -3.708398 51.49986 -52.55742 -47.76993      5
##
## $call
## searchK(documents = out$documents, vocab = out$vocab, K = c(3,
##     4, 5), init.type = "Spectral", N = 7, proportion = 0.5, heldout.seed = 1234,
##     M = 10, cores = 1, prevalence = ~ratings + regions + purchase_location +
##         user_type, max.em.its = 75, data = out$meta, verbose = FALSE)
##
## attr(,"class")
## [1] "searchK"
```

```r
plot(findingk_ver2.searchK)
```



The output of the `searchK` function provides various metrics for models with different numbers of topics (K). For each K (3, 4, 5), it provides:

1. **Exclusivity (exclus)**: This measures how unique the words within each topic are to that topic. The scores for exclusivity are quite similar for all Ks, which suggests that the topics generated are relatively distinct and exclusive for all three models.

2. **Semantic Coherence (semcoh)**: This measures how much the words within a topic co-occur in the same documents. The negative semantic coherence scores indicate relatively poor semantic coherence for all three models. The scores increase (become less negative) as K increases, indicating that the model's topics are becoming more semantically coherent as more topics are included.

3. **Held-out Likelihood (heldout)**: This measures how well the model predicts a held-out set of data. The held-out likelihood increases (becomes less negative) as K increases, which indicates that the model's predictive performance improves with more topics.

4. **Residual**: This is the difference between the observed values and the values predicted by the model. The residual decreases as K increases from 3 to 4 but then significantly increases when K equals 5, indicating that a 4-topic model might provide the best fit to the data among these options.

5. **Bound and Lower Bound (bound, lbound)**: These represent the upper and lower bounds of the variational approximation to the likelihood. The closer these two values are, the better the approximation. For all models, the bounds are quite close to each other, suggesting that the approximation is reasonable.

6. **Expectation-Maximization Iterations (em.its)**: This shows the number of expectation-maximization iterations that were run. The maximum number of iterations was reached for K = 3 and K = 4, but only 5 iterations were run for K = 5. This could indicate that the model with K = 5 did not converge, potentially due to overfitting or other issues.

Based on these results, it seems that a model with K = 4 provides the best balance between semantic coherence, exclusivity, and model fit t o t he d ata (as i ndicated b y t he r esidual a nd h eld-out likelihood). However, all the models seem to have relatively poor semantic coherence, suggesting that the topics might not be very meaningful or interpretable. This could be due to the simplicity and small size of the synthetic dataset. In a real-world scenario, you would typically have a much larger and more complex corpus of text data to analyze.

I have also added R code for doing STM on the hotel dataset in Canvas if you are interested.