3. Topic modeling

Thomas W. Jones

2018-04-29

Topic modeling

textmineR has extensive functionality for topic modeling. You can fit Latent Dirichlet Allocation (LDA), Correlated Topic Models (CTM), and Latent Semantic Analysis (LSA) from within textmineR. (Examples with LDA and LSA follow below.) As of this writing, textmineR's LDA and CTM functions are wrappers for other packages to facilitate a consistent workflow. (And textmineR takes advantage of the `RSpectra` package for LSA's single-value decomposition.) Plans exist to impelement LDA natively with `Rcpp` sometime in 2018.

textmineR's consistent representation of topic models boils down to two matrices. The first, "theta" (Θ), has rows representing a distribution of topics over documents. The second, phi (Φ), has rows representing a distribution of words over topics. In the case of probabilistic models, these are categorical probability distributions. For non-probabilistic models (e.g. LSA) these distributions are, obviously, not probabilities. With LSA, for example, there is a third object representing the sigular values in the decomposition.

In addition, textmineR has utility functions for topic models. This includes some original research. Examples include an R-squared for probabilistic topic models (working paper here), probabilistic coherence (a measure of topic quality), and a topic labeling function based on most-probable bigrams. Other utilities are demonstrated below

```
library(textmineR)

# load movie_review dataset from text2vec
data(movie_review, package = "text2vec")

str(movie_review)
#> 'data.frame':    5000 obs. of  3 variables:
#>  $ id       : chr  "5814_8" "2381_9" "7759_3" "3630_4" ...
#>  $ sentiment: int  1 1 0 0 1 1 0 0 0 1 ...
#>  $ review   : chr  "With all this stuff going down at the moment with MJ i've started
listening to his music, watching the odd docu"| __truncated__ "\\\"The Classic War of the
Worlds\\\" by Timothy Hines is a very entertaining film that obviously goes to great"|
__truncated__ "The film starts with a manager (Nicholas Bell) giving welcome investors (Robert
Carradine) to Primal Park . A s"| __truncated__ "It must be assumed that those who praised this
film (\\\"the greatest filmed opera ever,\\\" didn't I read some"| __truncated__ ...

# create a document term matrix
dtm <- CreateDtm(doc_vec = movie_review$review, # character vector of documents
                 doc_names = movie_review$id, # document names
                 ngram_window = c(1, 2), # minimum and maximum n-gram length
                 stopword_vec = c(tm::stopwords("english"), # stopwords from tm
                                  tm::stopwords("SMART")), # this is the default value
                 lower = TRUE, # lowercase - this is the default value
                 remove_punctuation = TRUE, # punctuation - this is the default
                 remove_numbers = TRUE, # numbers - this is the default
                 verbose = FALSE, # Turn off status bar for this demo
                 cpus = 2) # default is all available cpus on the system
```

LDA Example

To fit an LDA model in textmineR, use the `FitLdaModel` function. Input is a document term matrix. textmineR implements 2 methods for LDA, Gibbs sampling, and variational expectation maximization (also known as variational Bayes). The default is Gibbs sampling.

```
# start with a sample of 500 documents so our example doesn't take too long
dtm_sample <- dtm[ sample(1:nrow(dtm), 500) , ]

# Fit a Latent Dirichlet Allocation model
# note the number of topics is arbitrary here
# see extensions for more info
model <- FitLdaModel(dtm = dtm_sample,
                     k = 100,
                     iterations = 500,
                     alpha = 0.1, # this is the default value
                     beta = 0.05, # this is the default value
                     cpus = 2)
```

The output from the model is a list with the two matrices listed above.

```
# two matrices:
# theta = P(topic | document)
# phi = P(word | topic)
str(model)
#> List of 2
#>  $ theta: num [1:500, 1:100] 6.54e-07 1.15e-06 3.97e-02 7.63e-03 1.03e-02 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:500] "2205_2" "9985_1" "8151_10" "8561_10" ...
#>   .. ..$ : chr [1:100] "t_1" "t_2" "t_3" "t_4" ...
#>  $ phi  : num [1:100, 1:424926] 2.48e-07 1.78e-07 2.05e-07 2.52e-07 3.64e-07 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:100] "t_1" "t_2" "t_3" "t_4" ...
#>   .. ..$ : chr [1:424926] "watching_static" "crossed_nervous" "injections" "roam_forest" ...
```

Once we have created a model, we need to evaluate it. For overall goodness of fit, textmineR has R-squared and log likelihood. R-squared is interpretable as the proportion of variability in the data explained by the model, as with linear regression. For a full derivation and explanation of properties. See the working paper, here.

The log likelihood has a more difficult interpretation. Though, as shown in the R-squared working paper, R-squared and log likelihood are highly correlated.

```
# R-squared
# - only works for probabilistic models like LDA and CTM
model$r2 <- CalcTopicModelR2(dtm = dtm_sample,
                             phi = model$phi,
                             theta = model$theta,
                             cpus = 2)

model$r2
#> [1] 0.1417377

# log Likelihood (does not consider the prior)
# - only works for probabilistic models like LDA and CTM
model$ll <- CalcLikelihood(dtm = dtm_sample,
                           phi = model$phi,
                           theta = model$theta,
                           cpus = 2)
```

```
model$ll
#> [1] -464811.9
```

Next, we turn our attention to topic quality. There are many "topic coherence" metrics available in the literature. For example, see this paper or this paper. textmineR implements a new topic coherence measure based on probability theory. (A formal write up of this metric will be included in my PhD dissertation, expected 2020.)

Probabilistic coherence measures how associated words are in a topic, controlling for statistical independence. For example, suppose you have a corpus of articles from the sports section of a newspaper. A topic with the words {sport, sports, ball, fan, athlete} would look great if you look at correlation, without correcting for independence. But we actually know that it's a terrible topic because the words are so frequent in this corpus as to be meaningless. In other words, they are highly correlated with each other but they are statistically-independent of each other.

For each pair of words $\{a, b\}$ in the top M words in a topic, probabilistic coherence calculates $P(b|a) - P(b)$, where $\{a\}$ is more probable than $\{b\}$ in the topic.

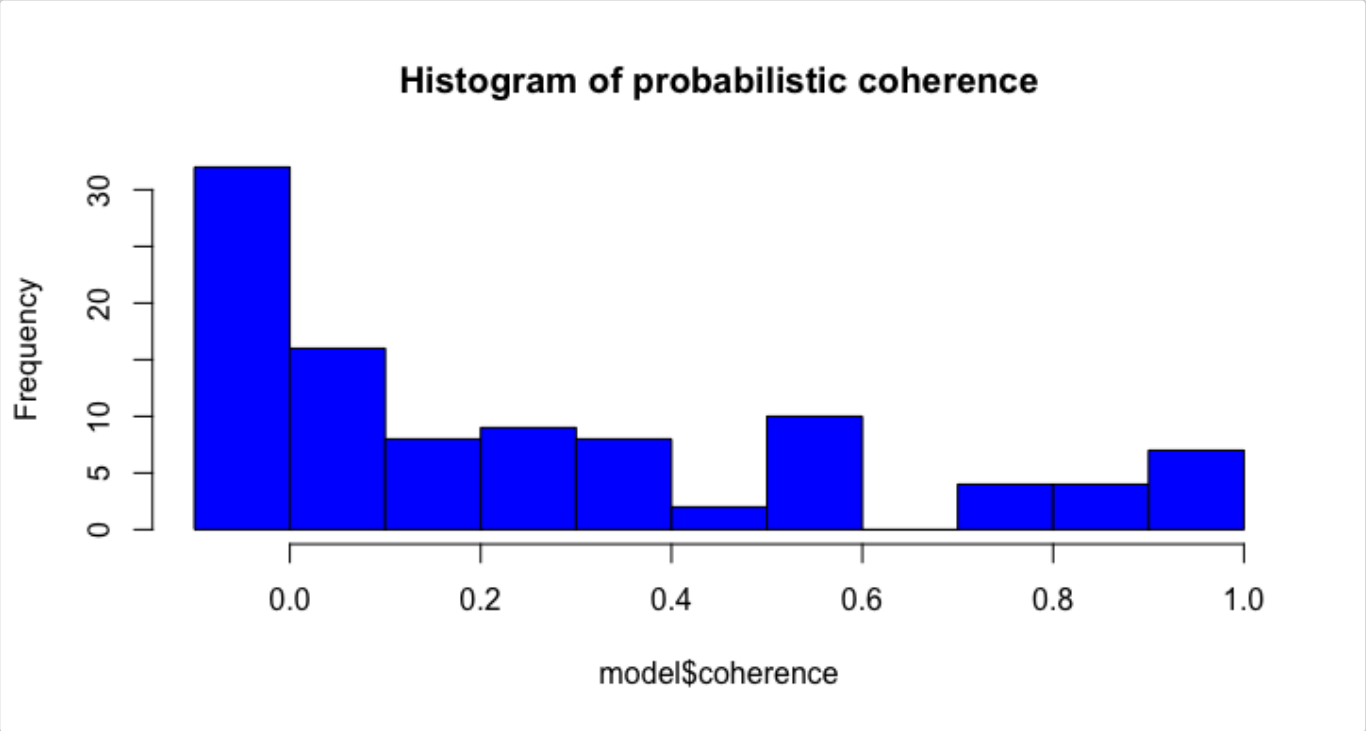
Here's the logic: if we restrict our search to only documents that contain the word $\{a\}$, then the word $\{b\}$ should be more more probable in those documents than if chosen at random from the corpus. $P(b|a)$ measures how probable $\{b\}$ is only in documents containing $\{a\}$. $P(b)$ measures how probable $\{b\}$ is in the corpus as a whole. If $\{b\}$ is not more probable in documents containing $\{a\}$, then the difference $P(b|a) - P(b)$ should be close to zero.

For example, suppose the top 4 words in a topic are $\{a, b, c, d\}$. Then, we calculate

1. $P(a|b) - P(b)$, $P(a|c) - P(c)$, $P(a|d) - P(d)$
2. $P(b|c) - P(c)$, $P(b|d) - P(d)$
3. $P(c|d) - P(d)$

And all 6 differences are averaged together, giving the probabilistic coherence measure.

```
# probabilistic coherence, a measure of topic quality
# this measure can be used with any topic model, not just probabilistic ones
model$coherence <- CalcProbCoherence(phi = model$phi, dtm = dtm_sample, M = 5)

summary(model$coherence)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> -0.0042 -0.0020  0.1706  0.2851  0.5085  0.9980

hist(model$coherence,
     col= "blue",
     main = "Histogram of probabilistic coherence")
```



We'll see the real value of coherence after calculating a few more objects. (Note: a future version of textmineR will calculate probabilistic coherence and the below objects automatically when you call `FitLdaModel` or any other topic model.)

In the chunk below, we will

1. Pull out the top 5 terms for each topic
2. Calculate the most frequent (prevalent) topics in the corpus
3. Get some bi-gram topic labels from a naive labeling algorithm (These naive labels are based on $P(\text{bi-gram}|\text{topic}) - P(\text{bi-gram})$. Noticing a theme?)

We'll then pull these together, along with coherence, into a table that summarizes the topic model.

```
# Get the top terms of each topic
model$top_terms <- GetTopTerms(phi = model$phi, M = 5)


head(t(model$top_terms)
```

t_1	boring_guy	kerr	crime_rate	siam	marry_boring
t_2	andrews	tierney	goodnik	bing_crosby	tom_tully
t_3	james_cole	mormon	brad_pitt	mormon_culture	jefferey
t_4	polanski	tenant	dench	roman_polanski	roman
t_5	caine_invisible	feminist	raised_movie	spoofs_film	wholly_convincing
t_6	romero	cotten	replaces	cate	mad_scientist

```
# Get the prevalence of each topic
# You can make this discrete by applying a threshold, say 0.05, for
# topics in/out of docuemnts.
model$prevalence <- colSums(model$theta) / sum(model$theta) * 100

# textmineR has a naive topic labeling tool based on probable bigrams
model$labels <- LabelTopics(assignments = model$theta > 0.05,
                            dtm = dtm_sample,
                            M = 1)
```

```
head(model$labels)
#>     label_1
#> t_1 "timothy_dalton"
#> t_2 "kristen_dunst"
#> t_3 "james_cole"
#> t_4 "br_br"
#> t_5 "production_quality"
#> t_6 "special_effects"

# put them together, with coherence into a summary table
model$summary <- data.frame(topic = rownames(model$phi),
                            label = model$labels,
                            coherence = round(model$coherence, 3),
                            prevalence = round(model$prevalence,3),
                            top_terms = apply(model$top_terms, 2, function(x){
                              paste(x, collapse = ", ")
                            }),
                            stringsAsFactors = FALSE)


model$summary[ order(model$summary$prevalence, decreasing = TRUE) , ][ 1:10 , ]
```

Summary of 10 most prevalent topics

	topic	label_1	coherence	prevalence	top_terms
t_89	t_89	watching_static	0.057	59.283	br, br_br, movie, film, good
t_15	t_15	dominick_eugene	0.297	0.714	dominick, eugene, novak, dominick_eugene, vertigo
t_81	t_81	br_br	0.296	0.704	match, lex, wwe, ladder, hart
t_72	t_72	south_street	0.543	0.651	widmark, skip, candy, peters, moe
t_59	t_59	red_shirt	0.098	0.615	noriko, shirt, bourne, kimiko, hoover
t_65	t_65	dan_real	0.298	0.601	anton, dandy, sergeant, macchesney, ballantine
t_35	t_35	science_fiction	0.197	0.587	modesty, puerto, tvm, puerto_rican, modesty_blaise
t_43	t_43	jet_li	0.391	0.583	jet_li, erika, li, jet, walter
t_99	t_99	larry_joe	0.933	0.579	olivier, oberon, richardson, mere, fog
t_46	t_46	nancy_drew	0.193	0.573	drew, nancy, nancy_drew, weird, mib

Ok, you've built a topic model. You've decided how well it fits your data. You've examined coherence, top words, and so on. Now you want to get topic distributions for new documents. (Remember, we only used 500 of our 5,000 documents to train the model.) To do this, we need Bayes' Rule.

The rows of Φ are $P(\text{word}|\text{topic})$. However, to get predictions for new documents, we need $P(\text{topic}|\text{word})$. Remembering Bayes' Rule, we get

$$P(\text{topic}|\text{word}) = \frac{P(\text{word}|\text{topic})P(\text{topic})}{P(\text{word})}$$

Detail-oriented readers may wonder how you can get $P(\text{topic})$. We can get this through $\sum_j P(\text{topic}|\text{document}_j)P(\text{document}_j)$.

For now, textmineR refers to the resulting matrix as Φ' or "phi prime". (Note: this will be called Γ or "gamma" in textmineR version 3.0+.)

textmineR's `CalcPhiPrime` function does the above calculations for you.

Once you have Φ', a simple dot product with the DTM of your new documents (A) will get new topic predictions.

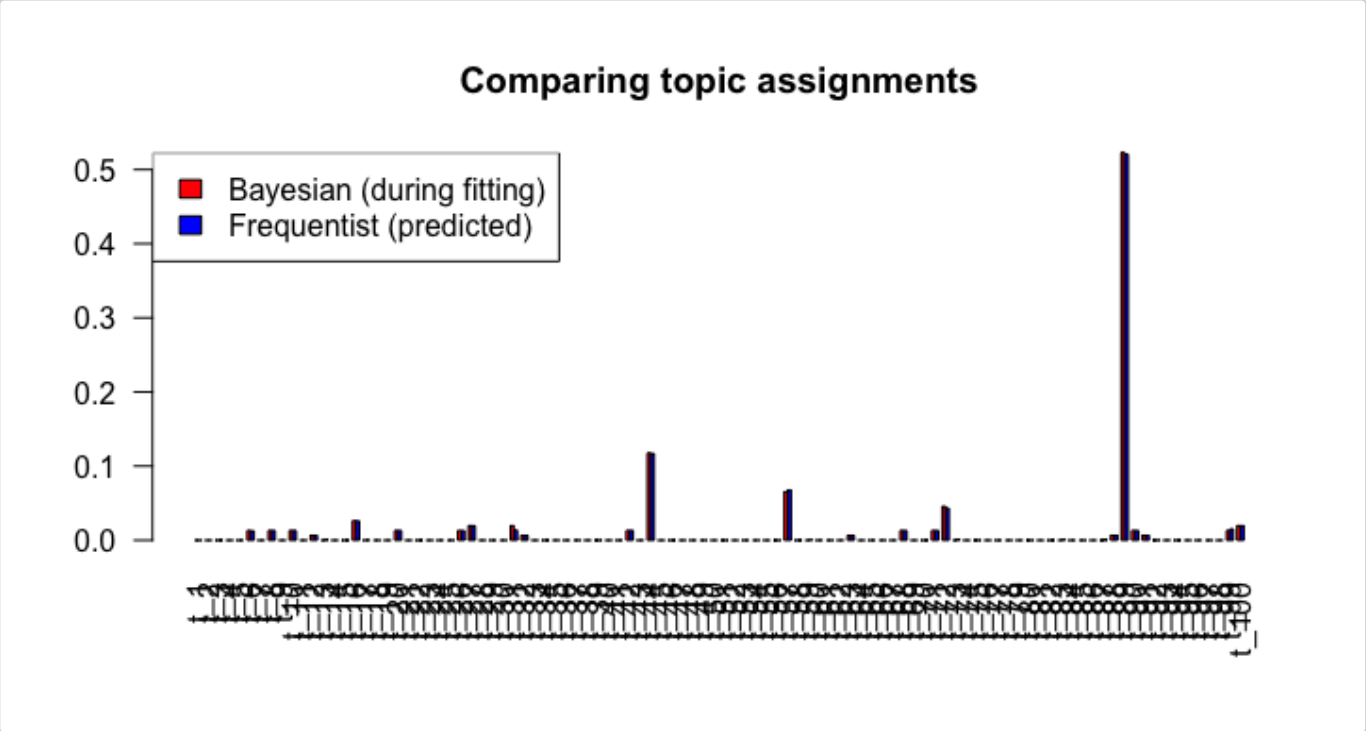
$$\Theta_{new} = A \cdot \Phi'^{T}$$

As of this writing, you will have to take care to make sure your vocabulary aligns. (I'd suggest using something like `intersect(colnames(dtm), colnames(theta))`.) (textmineR version 3.0 will enable a `predict` method for topic models that will handle all of this for you.)

```
# first get a prediction matrix, phi is P(word | topic)
# we need P(topic | word), or "phi_prime"
model$phi_prime <- CalcPhiPrime(phi = model$phi,
                                theta = model$theta)

# set up the assignments matrix and a simple dot product gives us predictions
assignments <- dtm / rowSums(dtm)

assignments <- assignments %*% t(model$phi_prime)

assignments <- as.matrix(assignments) # convert to regular R dense matrix
```

For the pedantic, the above method is a "frequentist" prediction even though LDA is a Bayesian model. textmineR 3.0 will implement a fully Bayesian predict method for LDA. In the meantime, this frequentist method works well, but is a little noisier. We can compare the difference between the frequentist predictions to the Bayesian ones, as the two barplots below show. By and large they are the same.

```
# compare the "fit" assignments to the predicted ones
barplot(rbind(model$theta[ rownames(dtm_sample)[ 1 ] , ],
              assignments[ rownames(dtm_sample)[ 1 ] , ]),
        las = 2,
        main = "Comparing topic assignments",
        beside = TRUE,
        col = c("red", "blue"))

legend("topleft",
       legend = c("Bayesian (during fitting)", "Frequentist (predicted)"),
       fill = c("red", "blue"))
```



Depending on your application, you can reformat the outputs of phi, theta, assignments, the summary table etc. to suite your needs. For example, you can build a "semantic" search of your documents by vectorizing the query with `CreateDtm`, then predicting under the model with phi_prime.

LSA Example

Latent semantic analysis was arguably the first topic model. LSA was patented in 1988. It uses a single value decomposition on a document term matrix, TF-IDF matrix, or similar.

In textmineR's notation:

$$A = \Theta \cdot S \cdot \Phi$$

Θ and Φ have the same (though non-probabilistic) interpretation as in LDA. S is the matrix of single values.

The workflow for LSA is largely the same for LDA. Two key differences: we will use the IDF vector mentioned above to create a TF-IDF matrix and we cannot get an R-squared for LSA as it is non-probabilistic.

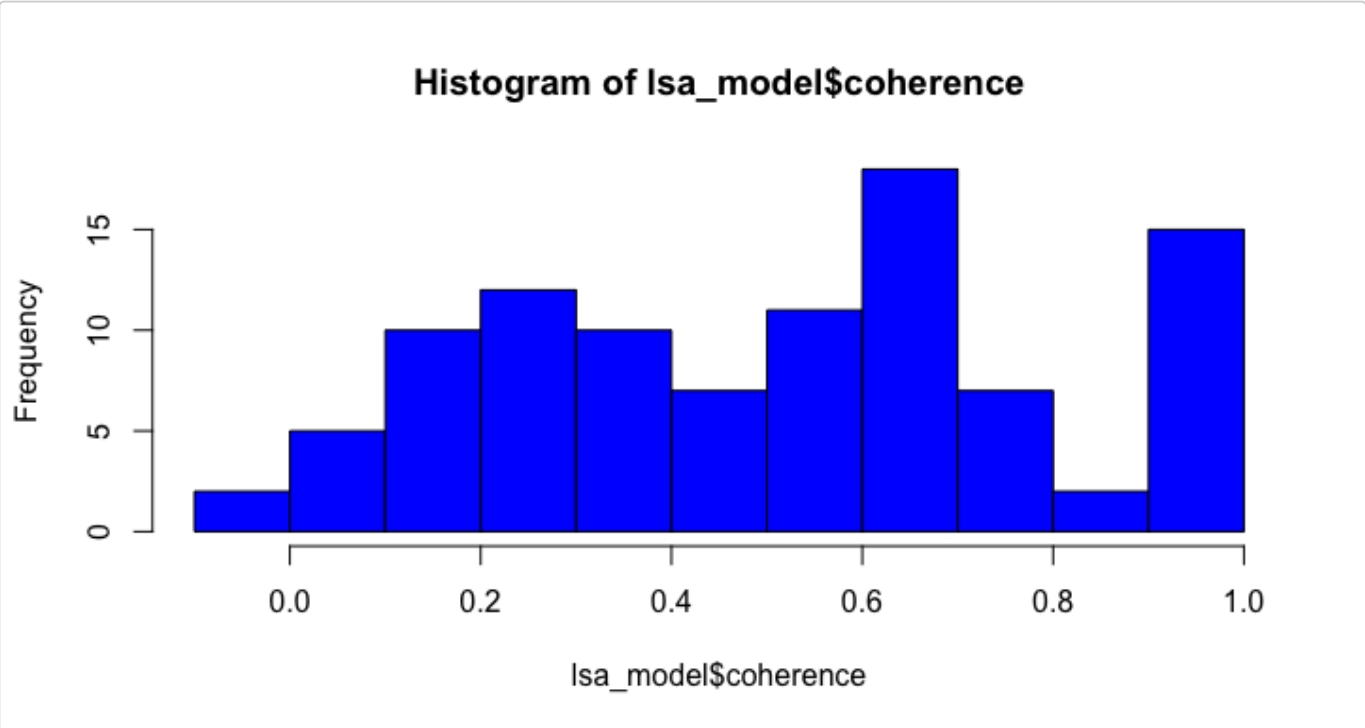
```
# get a tf-idf matrix
tf_sample <- TermDocFreq(dtm_sample)

tf_sample$idf[ is.infinite(tf_sample$idf) ] <- 0 # fix idf for missing words

tf_idf <- t(dtm_sample / rowSums(dtm_sample)) * tf_sample$idf

tf_idf <- t(tf_idf)

# Fit a Latent Semantic Analysis model
# note the number of topics is arbitrary here
# see extensions for more info
lsa_model <- FitLsaModel(dtm = tf_idf,
                     k = 100)

# three objects:
# theta = distribution of topics over documents
# phi = distribution of words over topics
# sv = a vector of singular values created with SVD
str(lsa_model)
#> List of 3
```

```
#>  $ sv   : num [1:100] 1.106 1.009 0.909 0.879 0.863 ...
#>  $ theta: num [1:500, 1:100] -0.00111 -0.00119 -0.00143 -0.00312 -0.0015 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:500] "2205_2" "9985_1" "8151_10" "8561_10" ...
#>   .. ..$ : chr [1:100] "t_1" "t_2" "t_3" "t_4" ...
#>  $ phi  : num [1:100, 1:424926] 0 0 0 0 0 0 0 0 0 0 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : chr [1:100] "t_1" "t_2" "t_3" "t_4" ...
#>   .. ..$ : chr [1:424926] "watching_static" "crossed_nervous" "injections" "roam_forest" ...
```

We cannot get a proper R-squared for an LSA model. (Actually, multiplying $\Phi \cdot S \cdot \Theta$ would give us exactly our document term matrix and an R-squared of 1. There isn't really a proper interpretation of $\Phi \cdot \Theta$ with LSA.)

However, we can still use probabilistic coherence to evaluate individual topics. We'll also get our top terms and make a summary table as we did with LDA, above.

```
# probabilistic coherence, a measure of topic quality
# - can be used with any topic lsa_model, e.g. LSA
lsa_model$coherence <- CalcProbCoherence(phi = lsa_model$phi, dtm = dtm_sample, M = 5)

summary(lsa_model$coherence)
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
#> -0.007146  0.250921  0.546600  0.512791  0.678000  0.998000         1

hist(lsa_model$coherence, col= "blue")
```




```
# Get the top terms of each topic
lsa_model$top_terms <- GetTopTerms(phi = lsa_model$phi, M = 5)


head(t(lsa_model$top_terms))
```

t_1	watching_static	crossed_nervous	injections	roam_forest	lead_protagonist
t_2	thought_quiet	watch_outtakes	quiet_good	outtakes_end	outtakes

t_3	br	movie	film	br_br	movies
t_4	movie_masterpieces	happiness_alienation	corruption_great	materialism_honor	mr_antonioni
t_5	eddie_murphy	murphy	eddie	stupid_animals	animals_people
t_6	eddie_murphy	eddie	murphy	stupid_animals	animals_people

```
# Get the prevalence of each topic
# You can make this discrete by applying a threshold, say 0.05, for
# topics in/out of docuemnts.
lsa_model$prevalence <- colSums(lsa_model$theta) / sum(lsa_model$theta) * 100

# textmineR has a naive topic labeling tool based on probable bigrams
lsa_model$labels <- LabelTopics(assignments = lsa_model$theta > 0.05,
                                dtm = dtm_sample,
                                M = 1)


head(lsa_model$labels)
```

	label_1
t_1	watching_static
t_2	thought_quiet
t_3	years_ago
t_4	movie_masterpieces
t_5	dental_work
t_6	eddie_murphy

```
# put them together, with coherence into a summary table
lsa_model$summary <- data.frame(topic = rownames(lsa_model$phi),
                                label = lsa_model$labels,
                                coherence = round(lsa_model$coherence, 3),
                                prevalence = round(lsa_model$prevalence,3),
                                top_terms = apply(lsa_model$top_terms, 2, function(x){
                                  paste(x, collapse = ", ")
                                }),
                                stringsAsFactors = FALSE)


lsa_model$summary[ order(lsa_model$summary$prevalence, decreasing = TRUE) , ][ 1:10 , ]
```

Summary of 10 most prevalent LSA topics

	topic	label_1	coherence	prevalence	top_terms
t_3	t_3	years_ago	0.058	71.137	br, movie, film, br_br, movies
t_2	t_2	thought_quiet	0.998	20.943	thought_quiet, watch_outtakes, quiet_good, outtakes_end, outtakes
t_16	t_16	lucille_ball	0.310	18.467	lucille_ball, lucille, ball, sad, br
t_18	t_18	dental_work	0.598	15.456	br, future_acting, acting_sigourney, weaver_perfect, holes_great

	topic	label_1	coherence	prevalence	top_terms
t_10	t_10	dental_work	0.113	12.052	sequels, dental_work, dental, years, movie_years
t_19	t_19	dental_work	0.048	11.638	br, film, br_br, rate, dental_work
t_7	t_7	lucille_ball	0.642	11.503	finished, ages_damn, takes_days, angel_rocks, damn_people
t_17	t_17	base_totally	0.642	10.667	suggest, base_totally, dvds_preordered, girl_escapes, bying_dvds
t_30	t_30	toole_susannah	0.336	9.994	games, game, dialogue_predictable, written_steve, teaser_written
t_20	t_20	br_br	0.155	9.758	br, br_br, rate, supposed_entertaining, images_year

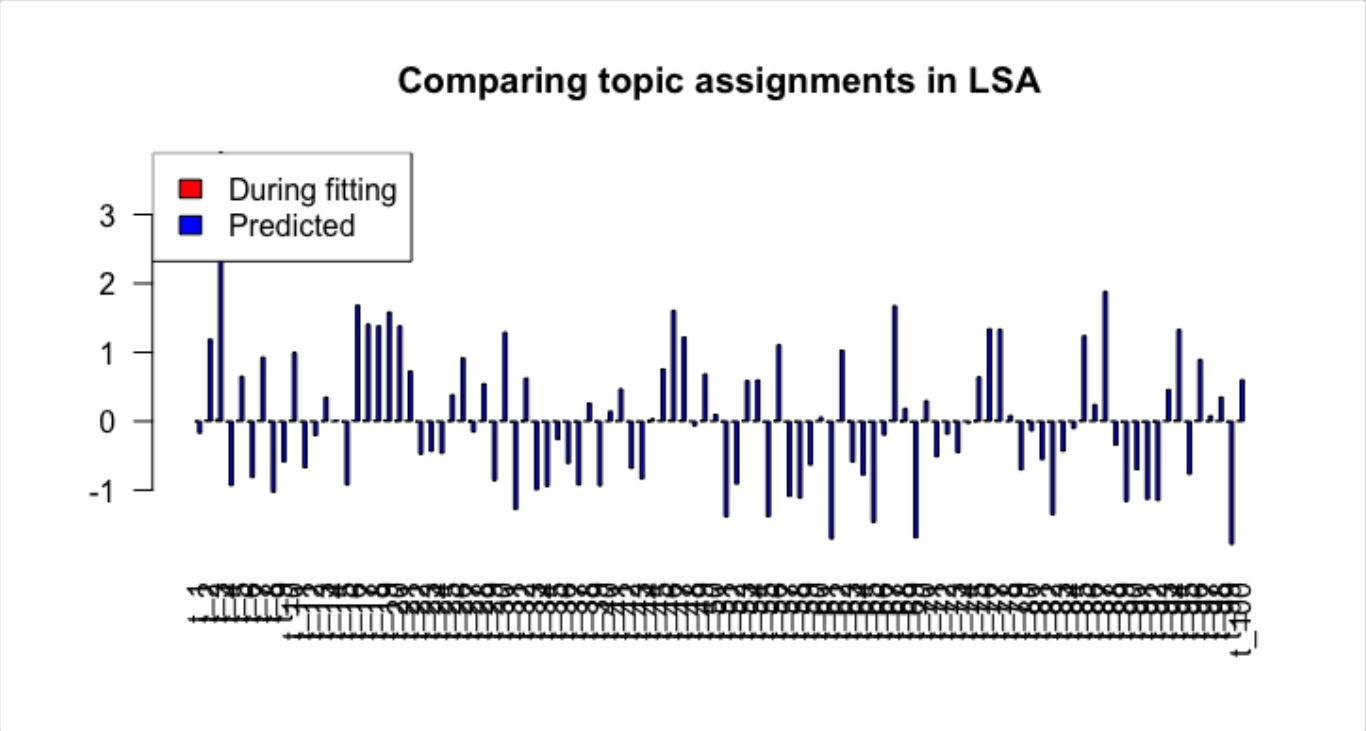
One key mathematical difference is how you calculate Φ'. For LSA the operation is

$$\Phi' = (S \cdot \Phi)^{-1}$$

```
# Get topic predictions for all 5,000 documents

# first get a prediction matrix,
lsa_model$phi_prime <- diag(lsa_model$sv) %*% lsa_model$phi

lsa_model$phi_prime <- t(MASS::ginv(lsa_model$phi_prime))

# set up the assignments matrix and a simple dot product gives us predictions
lsa_assignments <- t(dtm) * tf_sample$idf

lsa_assignments <- t(lsa_assignments)

lsa_assignments <- lsa_assignments %*% t(lsa_model$phi_prime)

lsa_assignments <- as.matrix(lsa_assignments) # convert to regular R dense matrix
```

In this case, there is no Bayesian/frequentist difference. So predictions are identical with both methods.

```
# compare the "fit" assignments to the predicted ones
barplot(rbind(lsa_model$theta[ rownames(dtm_sample)[ 1 ] , ],
              lsa_assignments[ rownames(dtm_sample)[ 1 ] , ]),
        las = 2,
        main = "Comparing topic assignments in LSA",
        beside = TRUE,
        col = c("red", "blue"))

legend("topleft",
       legend = c("During fitting", "Predicted"),
       fill = c("red", "blue"))
```



Other topic models

As of this writing, textmineR has implementations of

- LDA using Gibbs sampling from the `lda` package
- LDA using Bayesian expectation maximization from the `topicmodels` package
- LSA using a single value decomposition from the `RSpectra` package
- Correlated topic models (CTM) from the `topicmodels` package

A future version of textmineR will have an implementation of a strucural topic model from the `stm` package.

All of the above have nearly identical syntax and workflows as detailed above.

Extensions

Document clustering is just a special topic model

Document clustering can be thought of as a topic model where each document contains exactly one topic. textmineR's `Cluster2TopicModel` function allows you to take a clustering solution and a document term matrix and turn it into a probabilistic topic model representation. You can use many of textmineR's topic model utilities to evaluate your clusters (e.g. R-squared, coherence, labels, etc.)

Choosing the number of topics

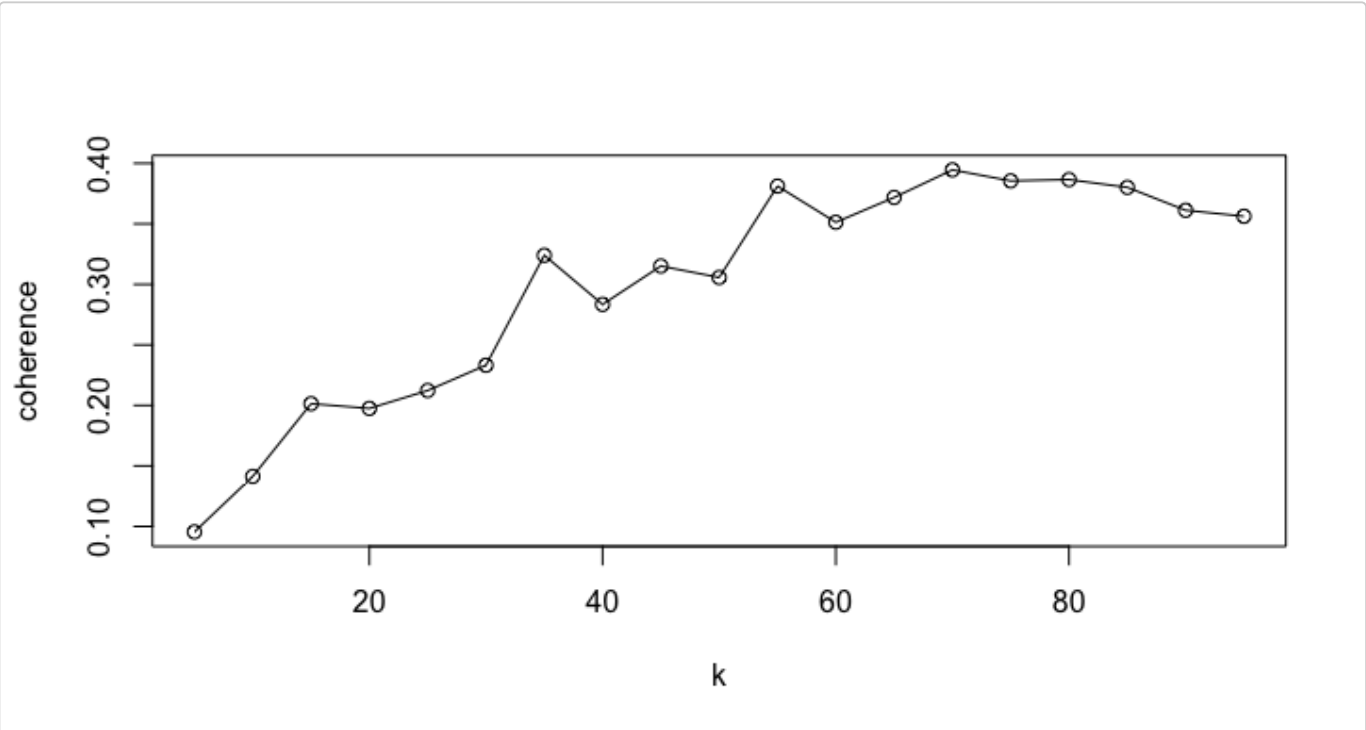
There is no commonly accepted way to choose the number of topics in a topic model. Fear not! Probabilistic coherence can help you. In forthcoming research, I show that probabilistic coherence can find the correct number of topics on a simulated corpus where the number of topics is known beforehand. (This will be part of a PhD dissertation, sometime around 2021. Stand by!)

Users can implement this procedure. Simply fit several topic models across a range of topics. Then calculate the probabilistic coherence for each topic in each model. Finally, average the probabilistic coherence across all topics in a model. This is similar to using the silhouette coefficient to select the number of clusters when clustering.

Some example code (on a trivially small dataset packaged with textmineR) is below.

```
# load a sample DTM
data(nih_sample_dtm)
```

```
# choose a range of k
# - here, the range runs into the corpus size. Not recommended for large corpora!
k_list <- seq(5, 95, by = 5)

# you may want toset up a temporary directory to store fit models so you get
# partial results if the process fails or times out. This is a trivial example,
# but with a decent sized corpus, the procedure can take hours or days,
# depending on the size of the data and complexity of the model.
# I suggest using the digest package to create a hash so that it's obvious this
# is a temporary directory
model_dir <- paste0("models_", digest::digest(colnames(nih_sample_dtm), algo = "sha1"))

# Fit a bunch of LDA models
# even on this trivial corpus, it will a bit of time to fit all of these models
model_list <- TmParallelApply(X = k_list, FUN = function(k){

  m <- FitLdaModel(dtm = nih_sample_dtm,
                   k = k,
                   iterations = 500,
                   cpus = 1)
  m$k <- k
  m$coherence <- CalcProbCoherence(phi = m$phi,
                                   dtm = nih_sample_dtm,
                                   M = 5)
  m
}, export=c("nih_sample_dtm"), # export only needed for Windows machines
cpus = 2)

# Get average coherence for each model
coherence_mat <- data.frame(k = sapply(model_list, function(x) nrow(x$phi)),
                            coherence = sapply(model_list, function(x) mean(x$coherence)),
                            stringsAsFactors = FALSE)


# Plot the result
# On larger (~1,000 or greater documents) corpora, you will usually get a clear peak
plot(coherence_mat, type = "o")
```



Using topic models from other packages

Topic models from other packages can be used with textmineR. The workflow would look something like this:

1. Use `CreateDtm` to create a curated DTM
2. Use `Dtm2Docs` to re-create a text vector of curated tokens from your DTM
3. Fit a topic model using your desired package (for example, mallet)
4. Format the raw output to have two matrices, phi and theta as above
5. Use textmineR's suite of utility functions with your model