Text as Data: Representation and Descriptive Inference

PS690 Computational Methods in Social Science

Jiawei Fu
Department of Political Science
Duke University

October 16, 2025

Overview

- We introduce some fundamentals of text analysis that predate the deep learning era.
- In particular, we will cover text representation, descriptive inference, and topic modeling — which remain useful even today.
- For other tasks like classification and sentiment analysis, DL methods are now the standard, so we will not cover older approaches such as Naive Bayes.
- You will learn more powerful techniques once we get to the DL section.

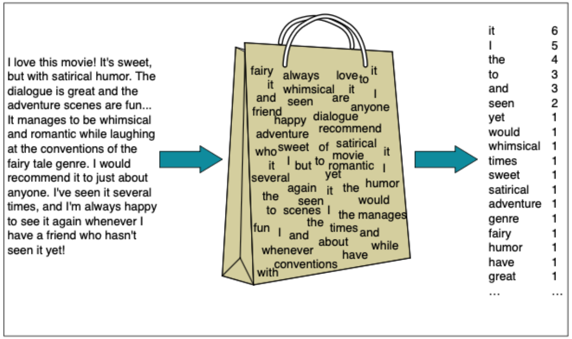
Overview

Terminology

- A **corpus** is a collection of **texts** that you want to analyze.
- The corpus can be anything from a few dozen newspaper articles to millions of tweets.
- Typically, the unit of analysis is the **document** within the corpus.
  - For example, if your corpus is all New York Times articles in 2020, then each article is a document.
  - If your corpus is legislative debates, then each speech by a legislator might be a document.

Representation

- The first challenge is to convert the words into a numerical representation that is suitable for use as the input to a deep neural network or other ML models.
- In general, we need map from raw text D to a numerical array C.
- The first idea is treat (text) document as a bag of words.



I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

Representation

- In bag of words, we represent each document by counting how many times each word appears in it.
- Suppose you have three documents:
  Doc1: "The cat sat on the mat"
  Doc2: "The dog barked loudly"
  Doc3: "The cat and the dog"

	cat	sat	mat	dog	barked	loudly	and	the
Doc1	1	1	1	0	0	0	0	2
Doc2	0	0	0	1	1	1	0	1
Doc3	1	0	0	1	0	0	1	2

- This is the document-feature matrix.
- This throws away a lot of information, like word order, but provides a parsimonious representation.

Tokenization

- To create the previous document-feature matrix, we have to break up a document into discrete units.
- Tokenization: splits the text into smaller constituent units, i.e. **token** (the smallest meaningful unit of a text).
- Each token is of a particular **type** (the name of the column).
- We will often refer to the set of types as the **vocabulary**.

	cat	sat	mat	dog	barked	loudly	and	the
Doc1	1	1	1	0	0	0	0	2
Doc2	0	0	0	1	1	1	0	1
Doc3	1	0	0	1	0	0	1	2

Tokenization

- A natural option of each token is each English word. (But for other languages, there is no white space that help us to split).
- Sometimes, single word is too narrow. For example, white house, some meaning is lost when we tokenize it as white and house.
- An n-gram is a sequence of n words:
    - A 2-gram (which we'll call bigram) is a two-word sequence of words like The water, or water of.
    - A 3-gram (a trigram) is a three-word sequence of words like The water of, or water of Walden.
- We add an underscore between words as am n-gram.

	the_cat	cat_sat	sat_on	on_the	the_mat
Doc1	1	1	1	1	1

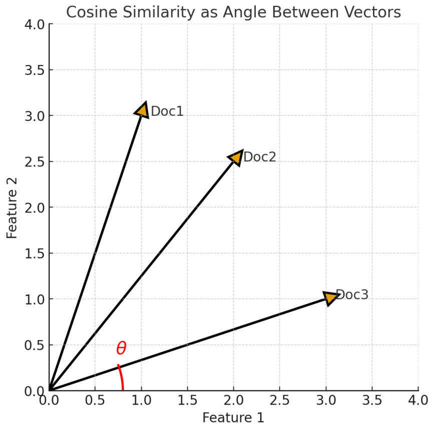
- Instead of including all n-grams, researchers might find it useful to retain a list of particular n-grams.

Tokenization

- For many applications, previous outputs are far too large.
- It is useful to further reduce the complexity after tonkenization that can reduce the dimension and do not hurt too much on the analysis.
- These steps include:
  1. Lowercase: replace all capital letters with lower letters.
  2. Remove punctuation.
  3. Remove stop words: common words used across documents that do not give much information about the task, like the, that, and.
  4. Lemmatize: In English, many words carry the same information due to a shared common root: like family, families, and family's. Lemma is the canonical form of a set of words that are related by inflection (modification due to case,number, tense, etc).
  5. Stem: A approximation to lemmatizing is stemming: we simply discard the end of a word using a few simple rules. For family, all of the variants are mapped to famili.

Text Similarity

- Each row of the document-feature matrix can be viewed as a vector in a high-dimensional space.
- Then, the similarity between two documents is equivalent to the "closeness" between two vectors.



Text Similarity

- One natural choice is looking at the angle between two vectors.

$$\begin{aligned}\text{cosine similarity}(d_1, d_2) &= \frac{d_1}{||d_1||} \cdot \frac{d_2}{||d_2||} \\ &= \frac{\sum_{i=1} d_{1i} d_{2i}}{(\sqrt{\sum_{i=1} d_{1i}^2})(\sqrt{\sum_{i=1} d_{2i}^2})}\end{aligned}$$

  where d_i^T is a row vector in the document-feature matrix.
- The cosine of the angle provides a nice measure od similarity, which normalizes the measure between 0 and 1.
- And it capture the relative direction, not affected by magnitude (for example dot product is affected by the length of the vector).

Visualization

- A word cloud is a visual representation of text data.
- The size of each word reflects its frequency (or weight) in the corpus.

Figure: Credit to Michael Franz.

Figure: Credit to K. A. Wisniewski.

TF-IDF

- Sometimes, raw counts of words are misleading, and not that informative.
- High frequency does not mean high importance: for example, like common words among all documents.
- Greatest signals from words are those in the goldilocks region- neither too rare nor too frequent.
- We can achieve this by re-weighting raw word counts.
- One popular weighting scheme is TF-IDF: term frequency inverse document frequency weighting: $d_{ij}^{tf-idf} = d_{ij} * \log \frac{N}{1+n_j}$, where N it the number of documents in the corpus, and n_j is the number of documents that contain word j; therefore, $\log \frac{N}{1+n_j}$ is the IDF. Recall, d_{ij} is the TF (word count).
- The core idea is to prioritize the words that are highly frequent in the document, but rare in the corpus over all.

Semantic Meaning of Word

- As we mentioned before, DFM throws away many information, especially, the meaning of the word.
- But how can we learn the meaning of a word?
- One idea is that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.
- For example, suppose you didn't know the meaning of the word ongchoi (a recent borrowing from Cantonese):
  1. Ongchoi is delicious sauteed with garlic.
  2. Ongchoi is superb over rice.
  3. ...ongchoi leaves with salty sauces...
- And suppose that you had seen many of these context words in other contexts:
  1. ...spinach sauteed with garlic over rice...
  2. ...chard stems and leaves are delicious...
  3. ..collard greens and other salty leafy greens.
- I hope you can infer that Ongchoi is a leafy green similar to spinach, chard, and collard greens.

Embedding

- Vector semantics is the standard way to represent word meaning in NLP.
- The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived (in ways we'll see) from the distributions of word neighbors.
- We now introduce a more powerful word representation: **embeddings**, short dense vectors.
- Unlike the vectors we've seen so far, embeddings are short, with number of dimensions d ranging from $50 - 1000$.
- We focus on static embeddings (a given word always maps to the same embedding vector) here, and will learn more powerful dynamic or contextualized embeddings like BERT after we learn Deep Learning.

Embeddings

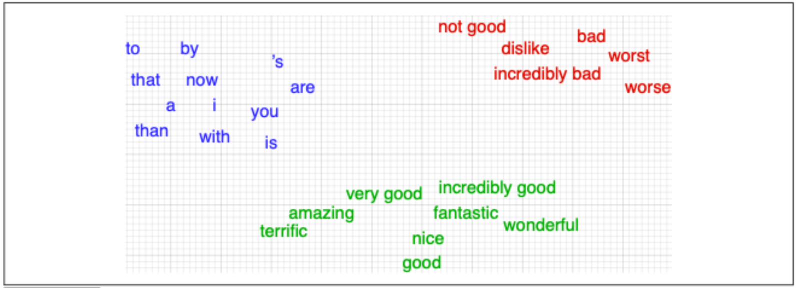


Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

- These embeddings capture elementary semantic properties, for example by mapping words with similar meanings to nearby locations in the embedding space.

Emneddings

- It turns out that the learned embedding space often has an even richer semantic structure than just the proximity of related words.
- For example, the concept that ‘Paris is to France as Rome is to Italy’ can be expressed through operations on the embedding vectors.

$$v(Paris) - v(France) + v(Italy) \approx v(Rome)$$

- Also: $v(King) - v(man) + v(woman) = v(queen)$

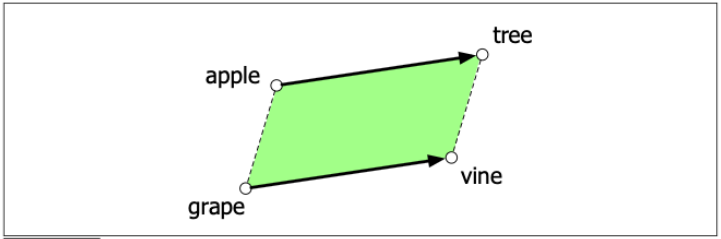


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

Embeddings

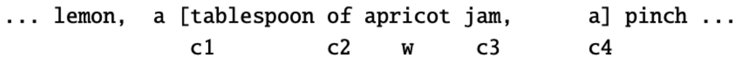
- Embeddings are short and dense, with number of dimensions d ranging from 50-1000, rather than the much larger vocabulary size $|V|$ we have seen.
- One method for computing embeddings: skip-gram with negative sampling.
- The skip-gram algorithm is one of two algorithms in a software package called word2vec, and so sometimes the algorithm is loosely referred to as word2vec.
- The intuition of word2vec: train a classifier on a binary prediction task: "Is word w likely to show up near Ongchoi?"
- We do not actually care about this prediction task; instead we will take the learned classifier weights as the word embeddings.

Embeddings

- The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier.
- A word c that occurs near the target word Ongchoi acts as gold 'correct answer' to the question "Is word c likely to show up near Ongchoi?"
- This method is called self-supervision, avoids the need for any sort of hand-labeled supervision signal.
- The intuition of skip-gram is:
    1. Treat the target word and a neighboring context word as positive examples.
    2. Randomly sample other words in the lexicon to get negative samples.
    3. Use logistic regression to train a classifier to distinguish those two cases.
    4. Use the learned weights as the embeddings.

Classifier

- Imagine a sentence like the following, with a target word apricot, and assume we're using a window of 2 context words:



```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2     w      c3      c4
```

Figure: [Jurafsky and Martin, 2025]

- Our goal is to train a classifier such that, given a tuple (w, c) of a target word w paired with a candidate context word c, it will return the probability that c is a real context word.
- For example, (w, c) is (apricot, jam), or perhaps (apricot, aardvark).
- We use $P(+|w, c)$ to denote the probability that c is the real context word.
- And $P(-|w, c) = 1 - P(+|w, c)$ is the probability that word c is not a real context word for w.

Classifier

- How does the classifier compute the probability P?
- The intuition of the skip-gram model is to base this probability on embedding similarity: a word is likely to occur near the target if its embedding vector is similar to the target embedding.
- Therefore, we can use dot product (after all, cosine is just a normalized dot product).
- Then, use logistic or sigmoid function to turn it into probability:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + exp(-c \cdot w)}$$

Word2vec

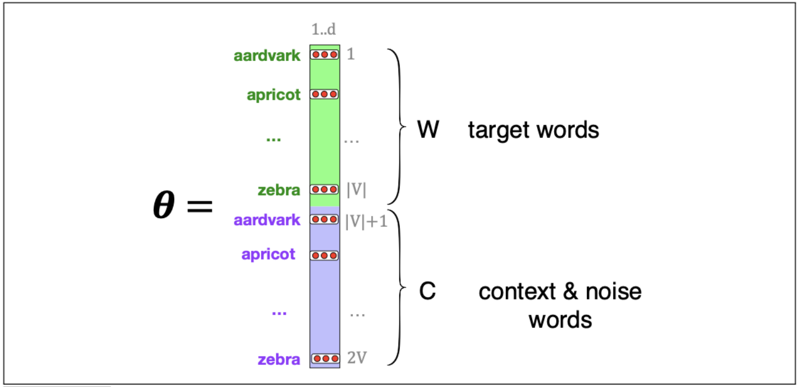
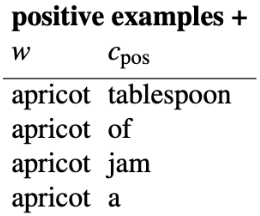


Figure 6.13 The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d, formed by concatenating two matrices, the target embeddings **W** and the context+noise embeddings **C**.

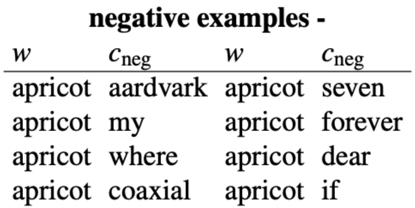
Word2vec

- The learning algorithm for skip-gram embeddings begins by assigning a random embedding vector for each of the N vocabulary words, and then proceeds to iteratively shift the embeddings.
- Next, prepare the data for the classifier.



positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a



negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

- Negative sampling create negative samples by randomly drawing word from the lexicon, constrained not to be the target word w.
- The noise words are chosen according to their weighted unigram frequency.

Word2vec

- If we consider one word/context pair (w, c_{pos}) with k noise words, the loss function is

$$\begin{aligned} L &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\ &= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right] \end{aligned}$$

Word2vec

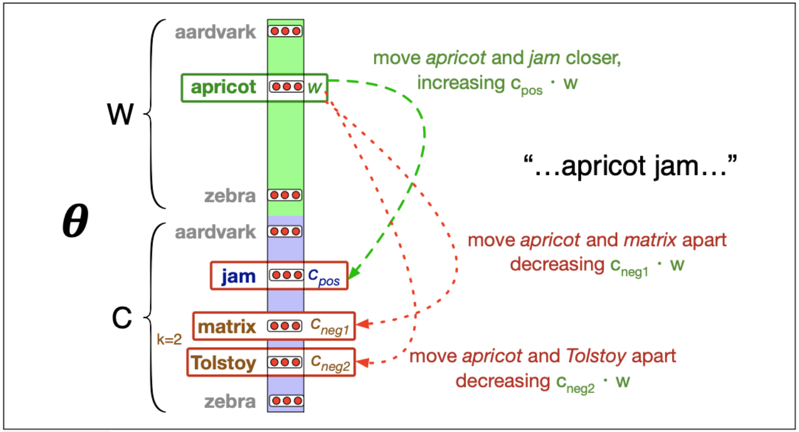
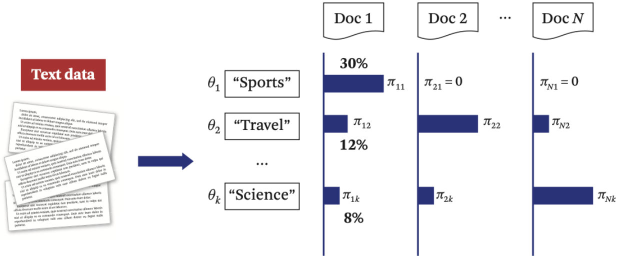

W
θ
C
aardvark
apricot
w
zebra
aardvark
jam
c_pos
matrix
c_neg1
k=2
Tolstoy
c_neg2
zebra
move apricot and jam closer, increasing c_pos · w
"…apricot jam…"
move apricot and matrix apart decreasing c_neg1 · w
move apricot and Tolstoy apart decreasing c_neg2 · w

Figure 6.14 Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

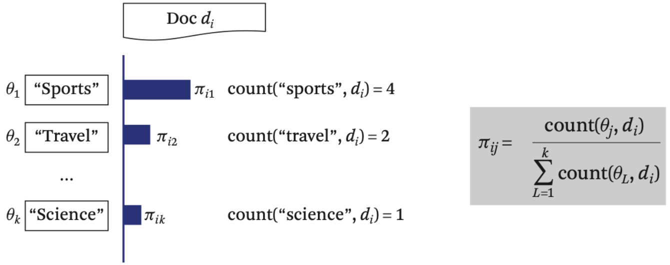
Topic Models



- Input:
  - a collection of N text documents $C = \{d_1, ..., d_N\}$
  - Number of topics: k
- Output:
  - k topics: $\{\theta_1, ..., \theta_k\}$
  - Coverage of topics in each d_i: $\pi_{i1}, ..., \pi_{ik}$ s.t. $\sum_{j=1}^{k} \pi_{ij} = 1$

Topics as Terms

- The simplest, natural way to define a topic is just as a term.
- A term can be a word or a phrase. For example, we may have terms like sports, travel, or science to denote three separate topics covered in text data.
- If we define a topic in this way, we can then analyze the coverage of such topics in each document based on the occurrences of these topical terms.



Doc d_i
θ_1 "Sports"
θ_2 "Travel"
...
θ_k "Science"
π_{i1} count("sports", d_i) = 4
π_{i2} count("travel", d_i) = 2
π_{ik} count("science", d_i) = 1

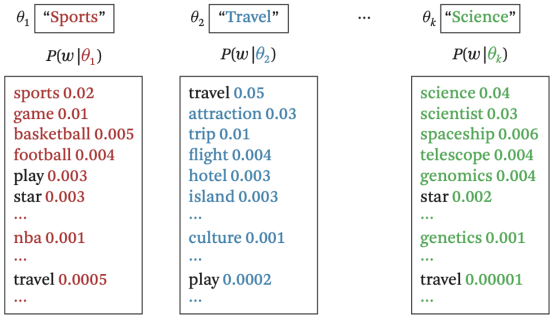
$$\pi_{ij} = \frac{\text{count}(\theta_j, d_i)}{\sum_{L=1}^{k} \text{count}(\theta_L, d_i)}$$

Figure: [Zhai and Massung, 2016]

Topics as Word Distributions

- Apparently, there are many problems for this simple method.
- When we count what words belong to a topic,we also need to consider related words.

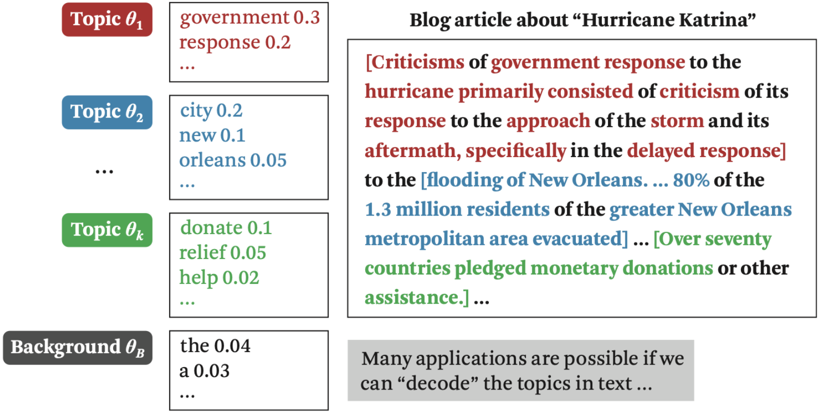
- A natural idea to address the problems of using one single term is to use more words to describe the topic.
- This can be achieved by a probability distribution over words.
- Note that $\sum_{w \in V} P(w|\theta_j) = 1$, where V is the set of vocabulary.



Probabilistic Latent Semantic Analysis (PLSA)

- We introduce probabilistic latent semantic analysis (PLSA), the most basic topic model, with many applications.
- As in all topic models, we make two key assumptions.
- First, we assume that a topic can be represented as a word distribution (or more generally a term distribution).
- Second, we assume that a text document is a sample of words drawn from a probabilistic model.

Probabilistic Latent Semantic Analysis (PLSA)



Blog article about "Hurricane Katrina"

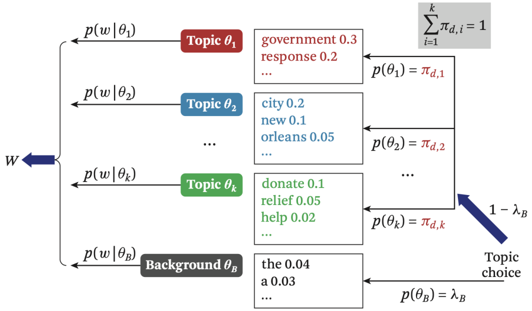
[Criticisms of government response to the hurricane primarily consisted of criticism of its response to the approach of the storm and its aftermath, specifically in the delayed response] to the [flooding of New Orleans. ... 80% of the 1.3 million residents of the greater New Orleans metropolitan area evacuated] ... [Over seventy countries pledged monetary donations or other assistance.] ...

Many applications are possible if we can "decode" the topics in text ...

Probabilistic Latent Semantic Analysis (PLSA)

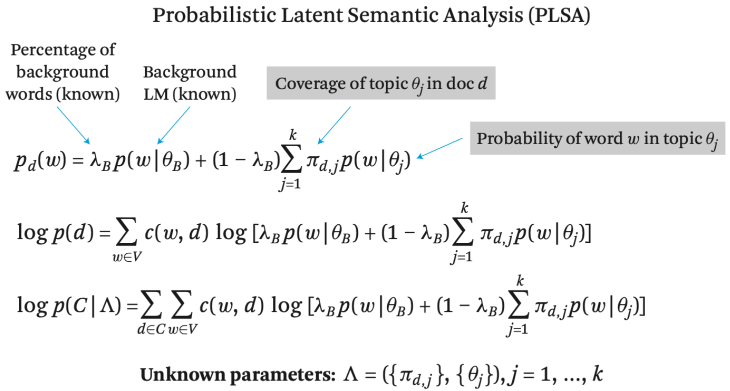
- PLSA is a generative model. First choose topics, then choose word.

- λ_B: the percentage of background words that we believe exist in the text data.
- $\pi_{d,j}$: the coverage of topic θ_j in document d.
- $p(w|\theta_k)$: word distribution of topic θ_k



Probabilistic Latent Semantic Analysis (PLSA)

- Now, we can write the likelihood function and estimate parameters by EM Algorithm.



Probabilistic Latent Semantic Analysis (PLSA)

Percentage of background words (known)
Background LM (known)
Coverage of topic θ_j in doc d
Probability of word w in topic θ_j

$$p_d(w) = \lambda_B p(w|\theta_B) + (1-\lambda_B)\sum_{j=1}^{k} \pi_{d,j}\, p(w|\theta_j)$$

$$\log p(d) = \sum_{w\in V} c(w,d) \log\left[\lambda_B p(w|\theta_B) + (1-\lambda_B)\sum_{j=1}^{k} \pi_{d,j}\, p(w|\theta_j)\right]$$

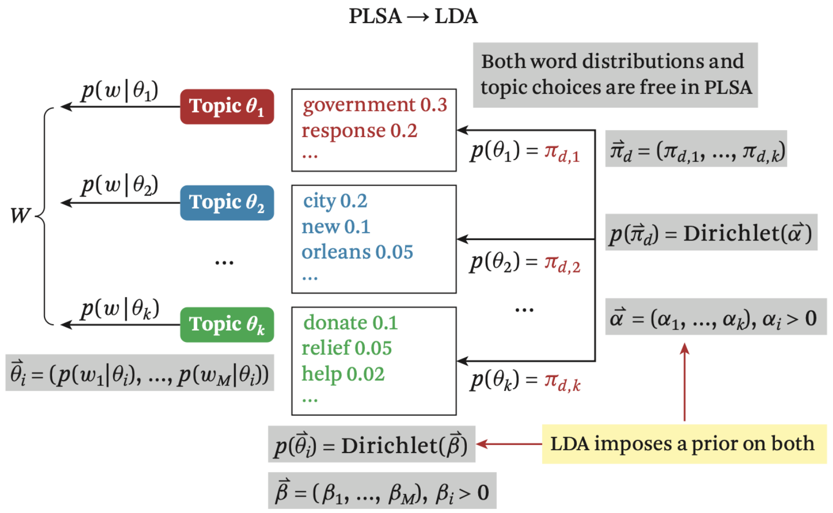
$$\log p(C|\Lambda) = \sum_{d\in C}\sum_{w\in V} c(w,d) \log\left[\lambda_B p(w|\theta_B) + (1-\lambda_B)\sum_{j=1}^{k} \pi_{d,j}\, p(w|\theta_j)\right]$$

Unknown parameters: $\Lambda = (\{\pi_{d,j}\}, \{\theta_j\}), j = 1, \ldots, k$

Latent Dirichlet Allocation

- We often have extra knowledge or our application imposes a particular preference for the topics to be analyzed.
- Also, PLSA is a generative model for modeling the words in a given document, but it is not a generative model for documents since it cannot give a probability of a new unseen document.
- Moreover, PLSA tends to overfit.
- One solution is to add priors on the parameters of PLSA and make a Bayesian version of the model.
- This leads to Latent Dirichlet Allocation (LDA).
- In LDA, the topic coverage distribution (a multinomial distribution) for each document is assumed to be drawn from a prior Dirichlet distribution.
- Similarly, all the word distributions representing the latent topics in a collection of text are also assumed to be drawn from another Dirichlet distribution.
- Thus, LDA only has parameters to characterize these two kinds of Dirichlet distributions.

Latent Dirichlet Allocation


PLSA → LDA
Both word distributions and topic choices are free in PLSA
$p(w|\theta_1)$
Topic θ_1
government 0.3
response 0.2
...
$p(\theta_1) = \pi_{d,1}$
$\vec{\pi}_d = (\pi_{d,1}, \ldots, \pi_{d,k})$
W
$p(w|\theta_2)$
Topic θ_2
city 0.2
new 0.1
orleans 0.05
...
$p(\vec{\pi}_d) = \text{Dirichlet}(\vec{\alpha})$
$p(\theta_2) = \pi_{d,2}$
...
$\vec{\alpha} = (\alpha_1, \ldots, \alpha_k), \alpha_i > 0$
$p(w|\theta_k)$
Topic θ_k
donate 0.1
relief 0.05
help 0.02
...
$p(\theta_k) = \pi_{d,k}$
$\vec{\theta}_i = (p(w_1|\theta_i), \ldots, p(w_M|\theta_i))$
$p(\vec{\theta}_i) = \text{Dirichlet}(\vec{\beta})$
LDA imposes a prior on both
$\vec{\beta} = (\beta_1, \ldots, \beta_M), \beta_i > 0$

Structural Topic Model

- STM is very similar to LDA, but it employs meta data about documents (such as the name of the author or the date in which the document was produced) to improve the assignment of words to latent topics in a corpus.
- For example, the θ for each document comes from a distribution that based on a vector of document covariates X_d

$$p(\pi) \sim LogisticNormal(\mu = X_d\gamma, \Sigma)$$

- Similarly, $p(\theta_i)$, distribution over words representing each topic, should also depends on some extra document-level content covariate.

References

Jurafsky, D. and Martin, J. H. (2025).
Speech and language processing 3rd edition draft.
Online Draft.

Zhai, C. and Massung, S. (2016).
Text data management and analysis: a practical introduction to information retrieval and text mining.
Morgan & Claypool.