CHAPTER .1

Topic Models

Imagine we have a corpus of restaurant reviews and want to learn what people are mostly talking about: what are their main complaints (price, quality, hygiene?), what types of restaurants (upscale, street food) are they discussing, and what cuisines (Italian, Asian, French, etc.) do they prefer? TF-IDF terms can give us individual words and phrases that hint at these things, but they do not group together into larger constructs. What we want is a way to automatically find these larger trends.

Topic models are one of the most successful and best-known applications of NLP in social sciences (not in small part due to its availability as a package in `R`). Their advantage is that they allow us to get a good high-level overview of the things that are being discussed in our corpus. What are the main threads, issues, and concerns in the texts? **Topic models**

What we ultimately want is a distribution over topics for each document. E.g., our first document is 80% topic 1, 12% topic 2, and 8% topic 3. In order to interpret these topics, we want for each topic a distribution over words. E.g., the five words most associated with topic 1 are "pasta", "red_wine", "pannacotta", "aglio", and "ragu". Based on these words, we can give it a descriptive label that sums up the words (here, for example "Italian cuisine"). This also gives us the basic elements: words, documents, and topics.

The most straightforward way to express the intuition above mathematically is through probabilities. The three elements can be related to each other in two conditional probability tables: one that tells us how likely each word is for each topic ($P(\text{word}|\text{topic})$), and one that tells us how likely each topic is for a specific document ($P(\text{topic}|\text{document})$). In practice, the distribution of words over topics is usually just called z, and the distribution over topics for each document is called θ. We alsi have a separate distribution θ_i for every document, rather than one that is shared for the entire corpus. After all, we do not expect that a document about a fast-food joint hits upon the same issues as a review of the swankiest 3-star Michelin place in town. Figure 1 shows the two quantities as graphical distributions.

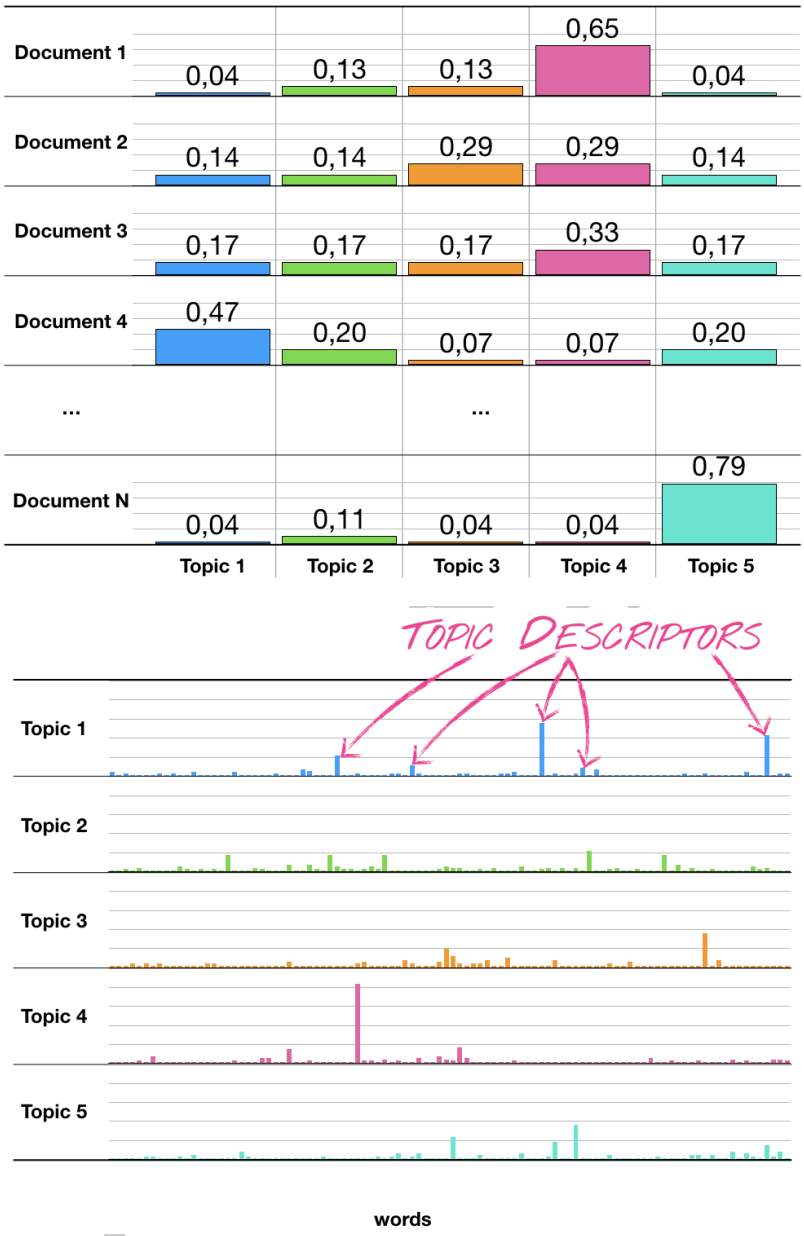

FIGURE 1. Graphical representation of the topic model parameters $\theta = P(\text{topic}|\text{document})$ (top) and $z = P(\text{word}|\text{topic})$ (bottom)

generative probabilistic models
Latent Dirichlet Allocation (LDA)

The term "topic model" covers a whole class of **generative probabilistic models**, with plenty of variations, but the most well-known is **Latent Dirichlet Allocation (LDA)** (Blei et al., 2003). So in the following, we will use the two terms interchangeably.

The operative word here is "generative": probabilistic models can be used to describe data, but they can just as well be used to *generate* data, as we have seen with language models. Topic models are very similar. In essence, a topic model is a way to imagine how a collection of documents was generated word by word. They have a Markov horizon of 0 (i.e., each word is independent from all the ones before), but instead condition each word on a topic. How this is supposed to have happened is described in the **generative process**. This is why LDA is a "generative model". The steps in this generative story for how a document came to be are fairly simple: for each word in a document, we first choose the topic of that slot, and then look up which word from the topic should fill the slot. Or, more precisely, for a document d_i:

generative process

(1) Set the number of words N by drawing from a **Poisson distribution**

Poisson distribution

(2) Draw a multinomial topic distribution θ over k topics from a **Dirichlet distribution**

Dirichlet distribution

(3) For $j \in N$:
   - Pick a topic t from the multinomial distribution θ
   - Choose word w_j based on its probability to occur in the topic according to the conditional probability $z_i j = P(w_j|t)$

No, this is of course not how documents are generated in real life, but it is a useful abstraction for our purposes. Because if we assume that this *is* indeed how the document was generated, then we can reverse-engineer the whole process to find the parameters that did it. And those are the ones we want. The generative story is often depicted in **plate notation** (Figure 2)

plate notation

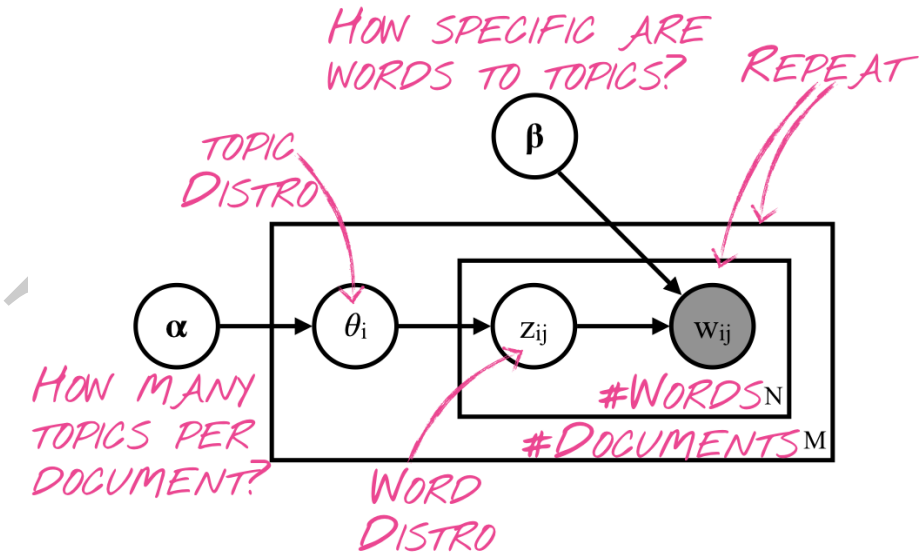

FIGURE 2. Plate notation of LDA model.

We have already seen θ and z. The first describes the probability of selecting a particular topic when sampling from a document. The second describes the chance of selecting a particular word when sampling from a given topic. Each of those probability distributions have a **hyperparameter** attached to them, α and β, respectively. α is the **topic prior**: it is the parameter for the Dirichlet process (a process that generates probability distributions) and controls how many topics we expect each document to have on average. If we expect every document to have several or all of the topics present, we choose an α value at 1.0 or above, leading to a rather uniform distribution. If we know or suspect that there are only a few topics per document (for example because the documents are quite short), we use a value smaller than 1.0, leading to a very peaked distribution. β is the **word prior**: it controls how topic-specific we expect the words to be. If we think that words are very specific to a particular topic, use a small β value (0.01 or less), if we think they are general, we use values up to 1.0

hyperparameter
topic prior
word prior

Now, we do not have the two tables yet, so we need to infer them from the data. This is usually done using either **Gibbs sampling** or **Expectation Maximization**. We do not have the space here to go into depth of how these algorithms work, but they are essentially guaranteed to find better and better parameter settings the longer we run them (i.e., the models become more and more likely to have produced the data we see). The question is of course: where do we start? In the first round, we essentially set all parameters to random values, and see what happens. Then we update the parameters in order to improve the fit with the data, and run again. We repeat this until we have reached a certain number of iterations, or the parameters stop changing much from iteration to iteration, i.e., the model fits the data more and more. We can measure the fit of the model to the data (also called the **data likelihood**) by multiplying together all the probabilities involved in generating the corpus.

Gibbs sampling
Expectation Maximization
data likelihood

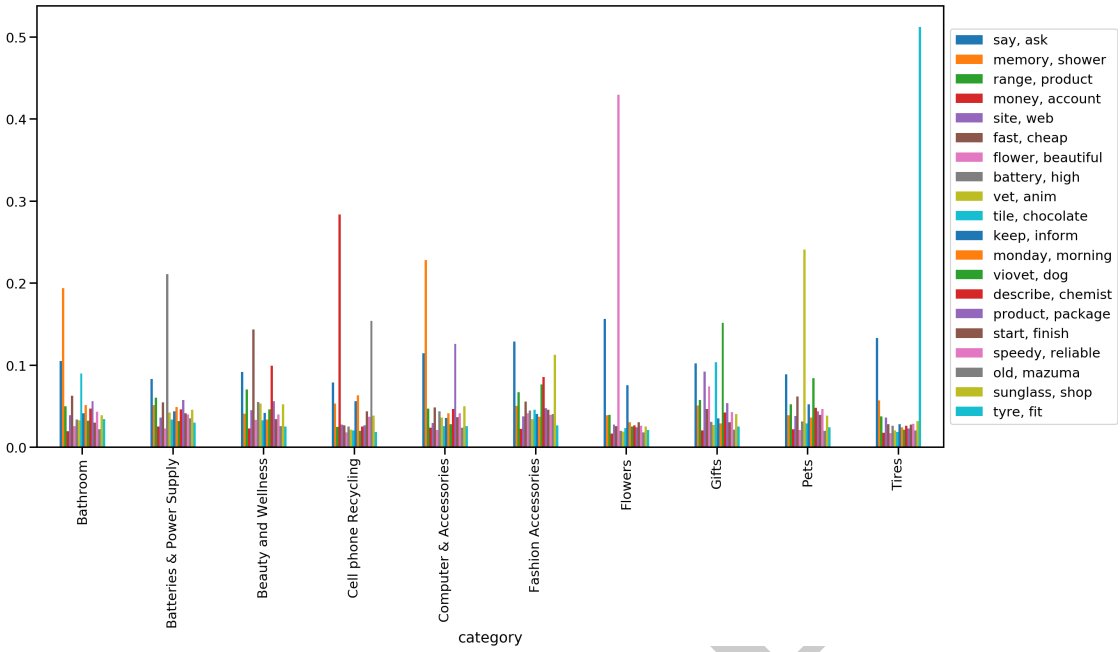

FIGURE 3. Distribution over 20 topics in 10 review categories

Running LDA on a topic of reviews can uncover some of the latent categories (see Figure 3). Here, we have 20 topics and only 10 underlying categories, but we allow for some slack. As we see, though, each category receives a dominant topic.

1. Selection and Evaluation

Because we initialize the models randomly, there is quite a lot of variation involved: if we run a topic model with the same number of topics 5 times on the same data, we will get 5 slightly different topic models. Some topics will be relatively stable between these models, but others might vary greatly. So how do we know which model to pick? The data likelihood of a model can also be used to calculate the **entropy** of the model, which in turn can be used to compute the **perplexity**. We can compare these numbers and select the model with the best perplexity. However, entropy and perplexity are model-inherent measures, i.e., they only depend on the parameters of the model. But what does that tell us about the interpretability of the model? In general, more likely models should also have more coherent and sensible topics, but that is a rough and tenuous equivalence. Which brings us to the question of **topic coherence**. Ideally, we want topics that are "self-explanatory", i.e., topics that are easy to interpret by anyone who sees them. An easy way to test this is through an **intrusion test**: take the top 5 words of a topic, replace one with a random word, and show this version to some people. If they can pick out the random intruder word, then the others must have been coherent enough to make it stick out. By asking several people and tracking how many guessed correctly, we can compute a score for each topic. Alternatively, we can use

entropy

perplexity

topic coherence

intrusion test

pointwise mutual information (PMI) for all pairs of the topic descriptor words, to see how likely we are to see these words together, as opposed to randomly distributed.

2. Caveats

The results depend strongly on the initialization and the parameter settings. There are a couple of best practices and rules of thumb (see also Boyd-Graber et al. (2014)):

(1) aggressively preprocess your data: use lemmatization, remove stopwords, replace numbers and user names, and join collocations.
(2) use a minimum document frequency of words: exclude all words that are rare (a word that we have seen 5 times in the data could just be a name or a common misspelling). Depending on your corpus size, a document frequency of 10, 20, 50, or even 100 is a good starting point.
(3) maximum document frequency of words: a word that occurs in more then 50% of all documents has very little discriminative power, so you might as well remove it. However, in order to get coherent topics, it can be good to go as low as 10%
(4) iterations: when is the model done fitting?

In either case, you will have to look at your topics and decide whether they make sense. This can be non-trivial: a topic might perfectly capture a topic you are not aware of (e.g., the words "BLEU, Bert, encoder, decoder, transformer" might look like random gibberish, but they perfectly encapsulate papers on machine translation).[1] Because topics are so variable from run to run, they are not stable indicators for dependent variables: we cannot use the output of a topic model to predict something like ratings.

In order to implement LDA in Python, there are two possibilities, `gensim` and `sklearn`. We will look at the `gensim` implementation here.

```
from gensim.models import LdaMulticore, TfidfModel
from gensim.corpora import Dictionary
import multiprocessing
```

CODE 1. Code for loading topic models from `gensim`.

```
dictionary = Dictionary(instances)
dictionary.filter_extremes(no_below=100, no_above=0.1)

ldacorpus = [dictionary.doc2bow(text) for text in instances]

tfidfmodel = TfidfModel(ldacorpus)

model_corpus = tfidfmodel[ldacorpus]
```

[1] Thanks to Hanh Hong Nguyen for the example.

CODE 2. Extracting and limiting the vocabulary, and transforming the data into TF-IDF representations of the vocabulary.

```
num_passes = 10
num_topics = 20
# find chunksize to make about 200 updates
chunk_size = len(model_corpus) * num_passes/200

model = LdaMulticore(model_corpus,
                     id2word=dictionary,
                     num_topics=num_topics,
                     workers=min(10, multiprocessing.cpu_count()-1),
                     passes=num_passes,
                     chunksize=chunk_size
                    )
```

CODE 3. Training the LDA model on the corpus on multiple cores.

```
import re

topic_corpus = model[model_corpus]

topic_sep = re.compile(r"0\.[0-9]{3}\*")
model_topics = [(topic_no, re.sub(topic_sep, '', model_topic).split('
    + ')) for topic_no, model_topic in
                model.print_topics(num_topics=num_topics, num_words
   =5)]

descriptors = []
for i, m in model_topics:
    print(i+1, ", ".join(m[:5]))
    descriptors.append(", ".join(m[:2]).replace('"', ''))
```

CODE 4. Enumerate the topics and print out the top 5 words for each.

3. Adding Constraints

Even when setting the hyperparameters well and having enough data to fit, there is a good chance that topics which we know should be separate can be merged by the model. One way to keep them apart is to tinker with z, by adding priors to some of the words. This will nudge the model towards some distributions and away from others. In practice, we are preventing the model from exploring some parameter configurations that we know will not lead to a solution we want. Jagarlamudi et al. (2012) introduced a straightforward implementation of such a **guided LDA**. In Python, it is available as a

guided LDA

new, separate library called `guidedlda`.

```
import numpy as np
import guidedlda
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(analyzer='word', ngram_range=(1,2), min_df=20, max_df=0.1, stop_words='english')
X = vectorizer.fit_transform(data)
vocab = vectorizer.get_feature_names()
word2id = dict((v, idx) for idx, v in enumerate(vocab))

indicators = {
    'PRICE': "price,expensive,cheap,cost,affordable,break bank".split(','),
    'SERVICE': "order,rude,customer service,staff,fast,service,management,wait,friendly,horrible service,slow".split(','),
    'AMBIENCE': "busy,lighting,decoration,ugly,cozy,noisy,loud,quiet".split(','),
    'PRODUCT': "delicious,quality,taste,ingredient,juicy,tasty,yummy,spicy".split(','),
    'HYGIENE': "clean,dirty,smelly,sparkling,kitchen,sink,toilet,floor".split(','),
    'FOOD': "burger,sandwich,pizza,burrito,hot dog,chili,bean,cheese,meat,ice cream".split(','),
    'CHAINS': "subway,mcdonald,burger king,chipotle,wendys,five guy,shake shack,".split(','),
    'OTHER': "menu,girlfriend,boyfriend,say,eat,drink,go".split(',')
}

seed_topic_list = [[w for w in words if w in set(vocab)] for words in indicators.values()]
for t, words in enumerate(seed_topic_list):
    print(topic_order[t], words)

model = guidedlda.GuidedLDA(n_topics=len(seed_topic_list), n_iter=1500, random_state=7, refresh=50, alpha=0.8, eta=0.000001)

seed_topics = {}
for t_id, st in enumerate(seed_topic_list):
    for word in st:
        seed_topics[word2id[word]] = t_id

# fit the model with seeds
doc_topic = model.fit_transform(X, seed_topics=seed_topics, seed_confidence=100)
```

```
34
35 n_top_words = 5
36 topic_word = model.topic_word_
37 for i, topic_dist in enumerate(topic_word):
38     topic_words = np.array(vocab)[np.argsort(topic_dist)][:-(
   n_top_words+1):-1]
39     print('Topic {}: "{}"'.format(topic_order[i], '" "'.join(
   topic_words)))
```

CODE 5. Using Guided LDA.

Bibliography

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

Jordan Boyd-Graber, David Mimno, and David Newman. 2014. *Care and Feeding of Topic Models: Problems, Diagnostics, and Improvements*, CRC Handbooks of Modern Statistical Methods. CRC Press, Boca Raton, Florida.

Jagadeesh Jagarlamudi, Hal Daumé III, and Raghavendra Udupa. 2012. Incorporating lexical priors into topic models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 204–213. Association for Computational Linguistics.