Topic modeling is a method for unsupervised classification of documents, similar to clustering on numeric data, which finds some natural groups of items (topics) even when we're not sure what we're looking for.

A document can be a part of multiple topics, kind of like in fuzzy clustering (soft clustering) in which each data point belongs to more than one cluster.

Why topic modeling?

Topic modeling provides methods for automatically organizing, understanding, searching, and summarizing large electronic archives.
It can help with:

- Discovering hidden themes in the collection.
- classifying the documents into the discovered themes.
- using the classification to organize/summarize/search the documents.

A document very often is about multiple topics and not just one clear-cut topic.

We can think of topics as clusters. The number of topics is a hyperparameter. The more topics we choose to have, the more likely our model is to overfit, because more topics means more specialization, more individualism for documents and differentiation from others which leads the model to find specifics and particularities of each document rather than more abstract topics.

The size of a cluster on the pyLDAvis graph indicates how many of the documents in our corpus align with the topic represented by that cluster. Smaller clusters imply that only a few documents talk about that specific topic, larger clusters mean that many of the documents talk about this specific topic.

Some clusters may overlap or be close to one another indicating that the topics are conceptually similar and therefore we need to decrease the number of topics hyperparameter.

Preprocessing Additional Steps:

Importance of bigrams and trigrams: Used mostly for datasets that heavily mention domain-specific terminology and technical jargon where a group of words is almost always used together to mean one thing

Bigrams and trigrams are groups of words such that when used together mean something different or more specific than when they are used individually.

Example on NeurIPS dataset: We often see the bigram term "neural network" being mentioned in these papers. If we perform usual NLP preprocessing "by the book", we will separate these 2 words from each other and the actual intended meaning is lost. Alone, "neural" can refer to anything that relates to neuron such as neuron, nervous system, nerve… and "network" can

refer to anything from a neural network to computer network to professional network… So, the words, when used individually, are removed from context and this can introduce noise into our clusters of topics. The solution is to treat the term "neural network" as one word: a bigram.
Example for a bigram: computer science, linear/dynamic programming…

Example for a trigram: object-oriented programming. These words strictly need to be clustered together to achieve the desired meaning.

How does LDA work?

There are 2 parts in LDA:

- The ***words that belong to a document***, this we already have and know.
- The ***words that belong to a topic*** or the probability of words belonging to a topic, this we need to calculate.

The Algorithm to find the latter:

- Go through each document and randomly assign each word in the document as belonging to one of ***k*** topics (*k* is the number of topics and is chosen beforehand as it is a hyperparameter).
- For each document ***d***, go through each word ***w*** and compute:

1. **P (topic *t* | document *d*)**: the **proportion of words in document *d* that have already been assigned to topic *t*.**
   Tries to capture how many words belong to a specific topic *t* for a given document *d* excluding the current word (because we want to use priors to decide which topic this current word should be assigned to). If a lot of words from document *d* belong to topic *t*, it is more probable that the current word *w* belongs to *t* as well.
   Formula: (# of words in *d* that belong to topic *t +alpha*/ total # of words in *d* (no matter which topic t they have been assigned to + *k***alpha*)
   N.B. Alpha and k*alpha are for smoothing

2. **P (word *w*| topic *t*)**: the proportion of assignments to topic *t* over all documents that come from this word *w*. Tries to capture how many documents belong to topic *t* because of the presence of word *w*.
   LDA represents documents as a mixture of topics. Similarly, a topic is a mixture of words. If a word has high probability of being in a topic, all the documents having *w* will be more strongly associated with *t* as well. Similarly, if *w* is not very probable to be in *t*, the documents which contain the *w* will be having very low probability of being in *t,* because rest of the words in *d* will belong to some other topic and hence *d* will have a higher probability for those topics. So even if *w* gets added to *t*, it won't be bringing many such documents to *t*.

Formula: P (word w | topic t) = (# of times the word w appears in topic t across all documents + β) / (# of words belonging to topic t across all documents + βV)

Where:

- V is the size of the vocabulary (i.e., the number of unique words in all the documents)
- β is a hyperparameter that controls the smoothing of the probability distribution

  The value of β is typically set to a small positive number to avoid division by zero and to ensure that each word has a nonzero probability of appearing in each topic.

3. Update the probability for the current word *w* belonging to topic *t* as:

   P (word w with topic t) = P (topic t | document d) * P (word w | topic t)

References: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2

Term Frequency-Inverse Document Frequency

Term Frequency Score TF = number of occurrences of a specific word in a document/total number of words in document

Inverse Document Frequency Score IDF = log (total number of documents in corpus/number of documents that contain the specific word)

TF-IDF = TF x IDF (range between 0 and 1)

The above number tells us how valuable/essential a certain document is for a certain word relative to all the other documents in the corpus. TF-IDF is proportional to the term frequency (i.e., words that appear very frequently in a document contribute to the document's topic more significantly) but inversely proportional to the document frequency so that words that appear very frequently across all documents are given less weight in determining the document's topic/main ideas. TF-IDF is proportional to TF and is offset by IDF.

The more frequent the usage of a word is across documents, the lower its IDF score (this is reflected in formula: if the denominator increases, the input to the log function decreases, so the IDF decreases). The lower the IDF score, the less important the word becomes. **Example for intuition:** It is very likely that the word 'the' will appear very frequently across all the documents in the corpus. This is in direct agreement with the fact that it is useless for determining the topic of a document.

TF-IDF results in a high weight for words that appear frequently in a particular document but are rare in the overall corpus, and a low weight for words that are common across many documents in the corpus.

Applications of TF-IDF: Search Engines.
Out of the millions of search results for a single search query inputted by a user, Google has to pick the result that is the most relevant to the query.