6 Tips for Interpretable Topic Models

Hands-on Python tutorial on tuning LDA topic models for easy-to-understand outputs.

Nicha Ruchirawat Nov 1, 2020 · 9 min read

With so much text outputted on digital platforms, the ability to automatically understand key topic trends can reveal tremendous insight. For example, businesses can benefit from understanding customer conversation trends around their brand and products. A common method to pick up key topics is Latent Dirichlet Allocation (LDA). However, outputs are often difficult to interpret for useful insights. We will explore techniques to enhance interpretability.

What is Latent Dirichlet Allocation (LDA)?

Latent Dirichlet Allocation (LDA) is a generative statistical model that helps pick up similarities across a collection of different data parts. In topic modeling, each data part is a word document (e.g. a single review on a product page) and the collection of documents is a corpus (e.g. all users' reviews for a product page). Similar sets of words occurring repeatedly may likely indicate topics.

LDA assumes that each **document is represented by a distribution of a fixed number of topics**, and **each topic is a distribution of words**.

Algorithm's high level key steps to approximate these distributions:

1. User select K, the number of topics present, tuned to fit each dataset.
2. Go through each document, and randomly assign each word to one of K topics. From this, we have a starting point for calculating **document distribution of topics p(topic t|document d)**, proportion of words in document d that are assigned to topic t. We can also calculate **topic distribution of words p(word w|topic t)**, proportion of word w in all documents' words that are assigned to topic t. These will be poor approximations due to randomness.
3. To improve approximations, we iterate through each document. For each document, go through each word and reassign a new topic, where we choose topic t with a probability **p(topic t|document d) *p(word w|topic t)** based on last round's distribution. This is essentially the **probability that topic t generated word w**. Recalculate p(topic t|document d) and p(word w|topic t) from these new assignments.
4. Keep iterating until topic/word assignments reach a steady state and no longer change much, (i.e. converge). Use final assignments to estimate topic mixtures of each document (%

words assigned to each topic within that document) and word associated to each topic (% times that word is assigned to each topic overall).

Data Preparation

We will explore techniques to optimize interpretability using LDA on Amazon Office Product reviews. To prepare the reviews data, we clean the reviews text with typical text cleaning steps:

1. Remove non-ascii characters, such as À µ ∅ ©
2. 'Lemmatize' words, which transform words to its most basic form, such as 'running' and 'ran' to 'run' so that they are recognized as the same word
3. Remove punctuation
4. Remove non-English comments if present

All code in the tutorial can be found here, where the functions for cleaning are located in clean_text.py. The main notebook for the whole process is topic_model.ipynb.

Steps to Optimize Interpretability

Tip #1: Identify phrases through n-grams and filter noun-type structures

We want to identify phrases so the topic model can recognize them. Bigrams are phrases containing 2 words e.g. 'social media'. Likewise, trigrams are phrases containing 3 words e.g. 'Proctor and Gamble'. There are many ways to detect n-grams, explained here. In this example, we will use *Pointwise Mutual Information (PMI) score.* This measures how much more likely the words co-occur than if they were independent. The metric is sensitive to rare combination of words, so it is used with an occurrence frequency filter to ensure phrase relevance. Bigram example below (trigram code included in Jupyter Notebook):

```
# Example for detecting bigrams
bigram_measures = nltk.collocations.BigramAssocMeasures()

finder =nltk.collocations.BigramCollocationFinder\
.from_documents([comment.split() for comment in\
clean_reviews.reviewText])

# Filter only those that occur at least 50 times
finder.apply_freq_filter(50)
bigram_scores = finder.score_ngrams(bigram_measures.pmi)
```

Additionally, we filter bigrams or trigrams with noun structures. This helps the LDA model better cluster topics, as nouns are better indicators of a topic being talked about. We use NLTK package to tag part of speech and filter these structures.

```
# Example filter for noun-type structures bigrams
def bigram_filter(bigram):
    tag = nltk.pos_tag(bigram)
    if tag[0][1] not in ['JJ', 'NN'] and tag[1][1] not in
['NN']:
        return False
    if bigram[0] in stop_word_list or bigram[1] in
stop_word_list:
        return False
    if 'n' in bigram or 't' in bigram:
        return False
    if 'PRON' in bigram:
        return False
    return True

# Can eyeball list and choose PMI threshold where n-grams stop
making sense
# In this case, get top 500 bigrams/trigrams with highest PMI
score
filtered_bigram = bigram_pmi[bigram_pmi.apply(lambda bigram:\
bigram_filter(bigram['bigram'])\
and bigram.pmi > 5, axis = 1)][:500]

bigrams = [' '.join(x) for x in filtered_bigram.bigram.values\
if len(x[0]) > 2 or len(x[1]) > 2]
```

Lastly, we concatenate these phrases together into one word.

```
def replace_ngram(x):
    for gram in bigrams:
        x = x.replace(gram, '_'.join(gram.split()))
    for gram in trigrams:
        x = x.replace(gram, '_'.join(gram.split()))
    return x

reviews_w_ngrams = clean_reviews.copy()
reviews_w_ngrams.reviewText = reviews_w_ngrams.reviewText\
.map(lambda x: replace_ngram(x))
```

Tip #2: Filter remaining words for nouns

In the sentence, ‘The store is nice’, we know the sentence is talking about ‘store’. The other words in the sentence provide more context and explanation about the topic (‘store’) itself. Therefore, filtering for noun extracts words that are more interpretable for the topic model.

An alternative is also to filter for both nouns ***and*** verbs.

```
# Tokenize reviews + remove stop words + remove names + remove
words with less than 2 characters
reviews_w_ngrams = reviews_w_ngrams.reviewText.map(lambda x:
[word for word in x.split()\
if word not in stop_word_list\
and word not in english_names\
and len(word) > 2])

# Filter for only nouns
def noun_only(x):
    pos_comment = nltk.pos_tag(x)
    filtered =[word[0] for word in pos_comment if word[1] in
['NN']]
    return filtered

final_reviews = reviews_w_ngrams.map(noun_only)
```

Tip #3: Optimize choice for number of topics through coherence measure

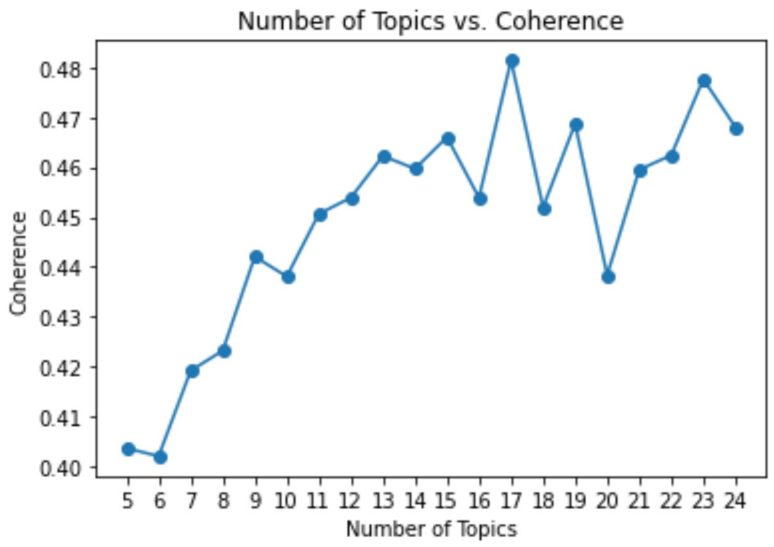
LDA requires specifying the number of topics. We can tune this through optimization of measures such as predictive likelihood, perplexity, and coherence. Much literature has indicated that maximizing a coherence measure, named Cv [1], leads to better human interpretability. We can test out a number of topics and asses the Cv measure:

```
coherence = []
for k in range(5,25):
    print('Round: '+str(k))
    Lda = gensim.models.ldamodel.LdaModel
    ldamodel = Lda(doc_term_matrix, num_topics=k, \
               id2word = dictionary, passes=40,\
                iterations=200, chunksize = 10000, eval_every =
None)

    cm = gensim.models.coherencemodel.CoherenceModel(\
         model=ldamodel, texts=final_reviews,\
         dictionary=dictionary, coherence='c_v')

    coherence.append((k,cm.get_coherence()))
```

Plotting this shows:



The improvement stops significantly improving after 15 topics. It is not always best where the highest Cv is, so we can try multiple to find the best result. We tried 15 and 23 here, and 23 yielded clearer results. Adding topics can help reveal further sub topics. Nonetheless, if the same words start to appear across multiple topics, the number of topics is too high.

Tip #4: Adjust LDA hyperparameters

```
Lda2 = gensim.models.ldamodel.LdaModel
```

```
ldamodel2 = Lda(doc_term_matrix, num_topics=23, id2word =
dictionary, passes=40,iterations=200,  chunksize = 10000,
eval_every = None, random_state=0)
```

If your topics still do not make sense, try increasing passes and iterations, while increasing chunksize to the extent your memory can handle.

`chunksize` is the number of documents to be loaded into memory each time for training. **`passes`** is the number of training iterations through the entire corpus. **`iterations`** is the maximum iterations over each document to reach convergence — limiting this means that some documents may not converge in time. If the training corpus has 200 documents, **`chunksize`** is 100, **`passes`** is 2, and **`iterations`** is 10, algorithm goes through these rounds:

Round #1: documents 0–99
Round #2: documents 100–199
Round #3: documents 0–99
Round #4: documents 100–199

Each round will iterate each document’s probability distribution assignments for a maximum of 10 times, moving to the next document before 10 times if it already reached convergence. This is basically algorithm’s key steps 2–4 explained earlier, repeated for the number of **`passes`**, while step 3 is repeated for 10 **`iterations`** or less.

The topic distributions for entire corpus is updated after each **`chunksize`**, and after each **`passes`**. Increasing **`chunksize`** to the extent your memory can handle will increase speed as topic distribution update is expensive. However, increasing `chunksize` requires increasing number of `passes` to ensure sufficient corpus topic distribution updates, especially in small corpuses. `iterations` also needs to be high enough to ensure a good amount of documents reach convergence before moving on. We can try increasing these parameters when topics still don't make sense, but logging can also help debug:

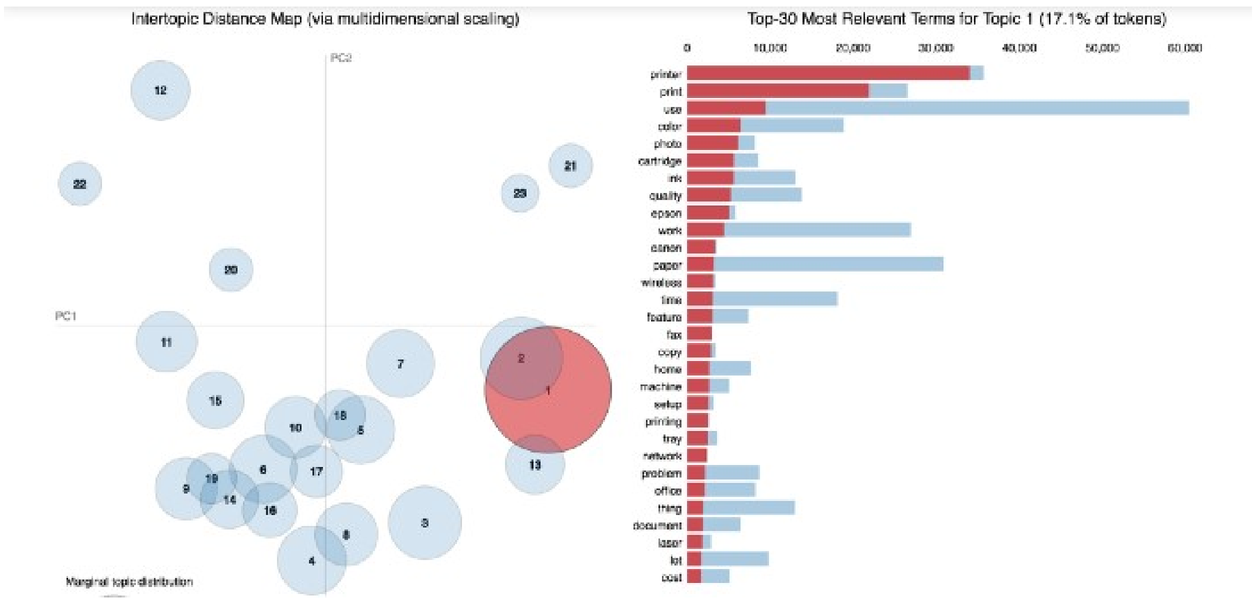
```
import logging
logging.basicConfig(filename='gensim.log',
                    format="%(asctime)s:%(levelname)s:%
(message)s",
                    level=logging.INFO)
```

Look for a lines that look like this in the log, which will repeat for the number of `passes` that you set:

```
2020-07-21 06:44:16,300 - gensim.models.ldamodel - DEBUG -
100/35600 documents converged within 200 iterations
```

By the end of the `passes`, most of the documents should have converged. If not, increase `passes` and `iterations`.

Tip #5: Use pyLDAvis to visualize topic relationships



The pyLDAvis [2] package in Python gives two important pieces of information. The circles represent each topic. The distance between the circles visualizes topic relatedness. These are mapped through dimensionality reduction (PCA/t-sne) on distances between each topic's probability distributions into 2D space. This shows whether our model developed distinct topics. We want to tune model parameters and number of topics to minimize circle overlap.

Topic distance also shows how related topics are. Topics 1,2,13 clustered together talk about electronics (printers, scanners, phone/fax). Topics in quadrant 3 such as 6,14,19 are about office stationary (packaging materials, post-its, file organizer). Additionally, circle size represents topic prevalence. For example, topic 1 makes up the biggest portion of topics being talked about amongst documents, constituting 17.1% of the tokens.

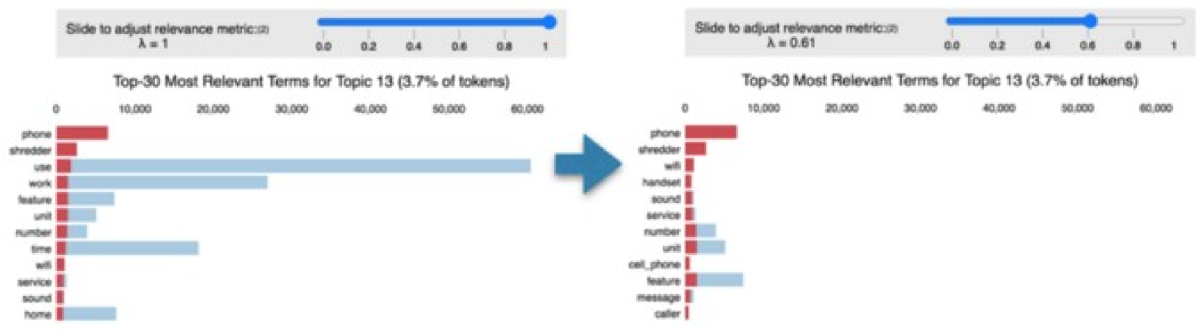
Tip #6: Tune relevancy score to prioritize terms more exclusive

to a topic

Words representing a given topic may be ranked high because they are globally frequent across a corpus. Relevancy score helps prioritize terms that belong more exclusively to a given topic, making the topic more obvious. The relevance of term w to topic k is defined as:

$$r(w, k|\lambda) = \lambda log(\phi_{kw}) + (1 - \lambda)log(\frac{\phi_{kw}}{p_{kw}})$$

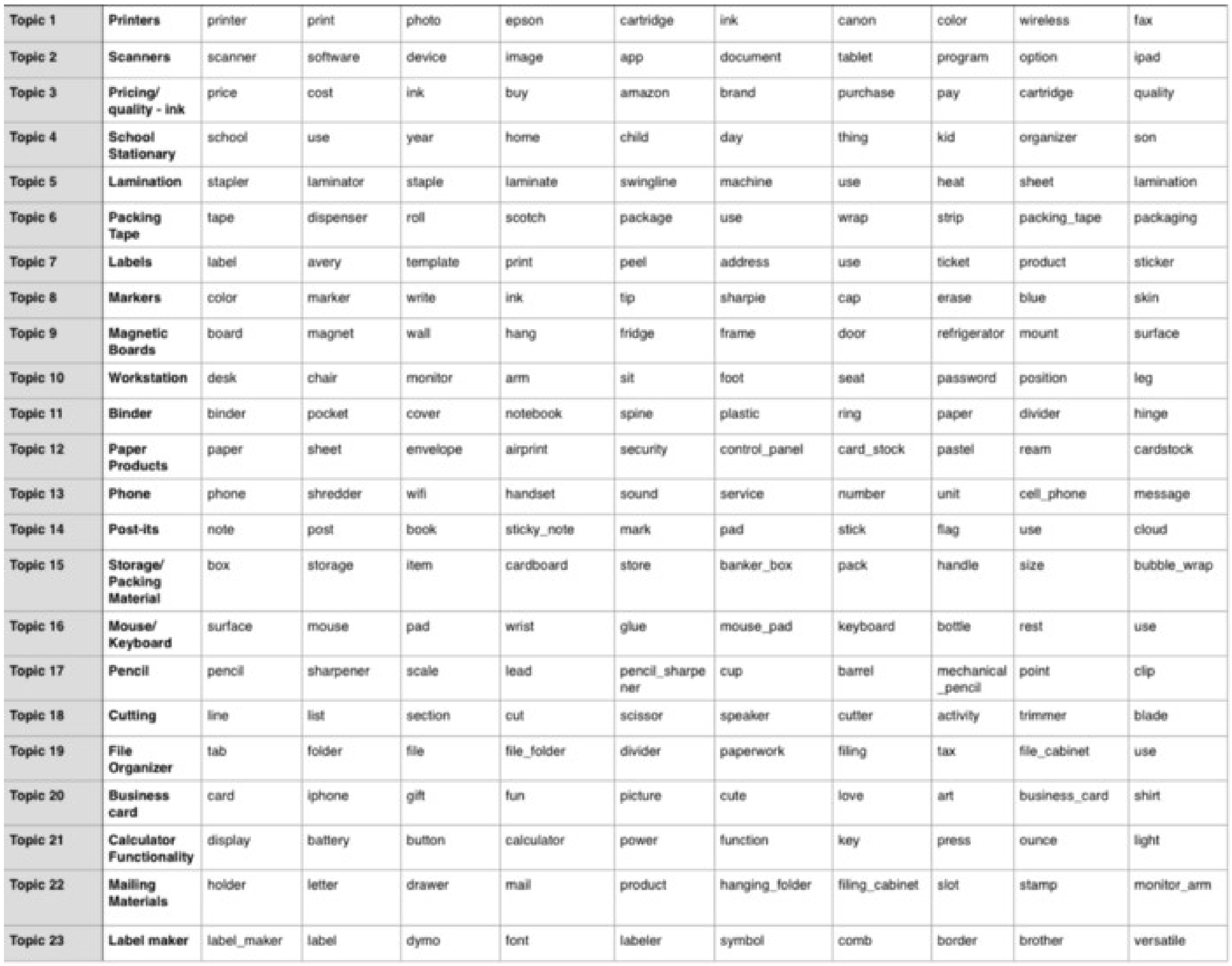
where ϕ_kw is the probability of word w in topic k and ϕ_kw/p_kw is the lift in term's probability within a topic to its marginal probability across the entire corpus (this helps discards globally frequent terms). A lower λ gives more importance to the second term (ϕ_kw/p_kw), which gives more importance to topic exclusivity. We can again use pyLDAvis for this. For instance, when lowering λ to 0.6, we can see that topic 13 ranked terms that are even more relevant to the topic of phones.



Dial the lambda around to get the result that makes the most sense and apply the optimal lambda value to obtain the output:

```
all_topics = {}
lambd = 0.6  # Adjust this accordingly
for i in range(1,22): #Adjust number of topics in final model
    topic = topic_data.topic_info[topic_data.topic_info\
            .Category == 'Topic'+str(i)]
    topic['relevance'] = topic['loglift']*(1-lambd)\
                         +topic['logprob']*lambd
    all_topics['Topic '+str(i)] =
topic.sort_values(by='relevance\
    , ascending=False).Term[:10].values
```

Final Results



Topic 1	**Printers**	printer	print	photo	epson	cartridge	ink	canon	color	wireless	fax
Topic 2	**Scanners**	scanner	software	device	image	app	document	tablet	program	option	ipad
Topic 3	**Pricing/ quality - ink**	price	cost	ink	buy	amazon	brand	purchase	pay	cartridge	quality
Topic 4	**School Stationary**	school	use	year	home	child	day	thing	kid	organizer	son
Topic 5	**Lamination**	stapler	laminator	staple	laminate	swingline	machine	use	heat	sheet	lamination
Topic 6	**Packing Tape**	tape	dispenser	roll	scotch	package	use	wrap	strip	packing_tape	packaging
Topic 7	**Labels**	label	avery	template	print	peel	address	use	ticket	product	sticker
Topic 8	**Markers**	color	marker	write	ink	tip	sharpie	cap	erase	blue	skin
Topic 9	**Magnetic Boards**	board	magnet	wall	hang	fridge	frame	door	refrigerator	mount	surface
Topic 10	**Workstation**	desk	chair	monitor	arm	sit	foot	seat	password	position	leg
Topic 11	**Binder**	binder	pocket	cover	notebook	spine	plastic	ring	paper	divider	hinge
Topic 12	**Paper Products**	paper	sheet	envelope	airprint	security	control_panel	card_stock	pastel	ream	cardstock
Topic 13	**Phone**	phone	shredder	wifi	handset	sound	service	number	unit	cell_phone	message
Topic 14	**Post-its**	note	post	book	sticky_note	mark	pad	stick	flag	use	cloud
Topic 15	**Storage/ Packing Material**	box	storage	item	cardboard	store	banker_box	pack	handle	size	bubble_wrap
Topic 16	**Mouse/ Keyboard**	surface	mouse	pad	wrist	glue	mouse_pad	keyboard	bottle	rest	use
Topic 17	**Pencil**	pencil	sharpener	scale	lead	pencil_sharpe ner	cup	barrel	mechanical _pencil	point	clip
Topic 18	**Cutting**	line	list	section	cut	scissor	speaker	cutter	activity	trimmer	blade
Topic 19	**File Organizer**	tab	folder	file	file_folder	divider	paperwork	filing	tax	file_cabinet	use
Topic 20	**Business card**	card	iphone	gift	fun	picture	cute	love	art	business_card	shirt
Topic 21	**Calculator Functionality**	display	battery	button	calculator	power	function	key	press	ounce	light
Topic 22	**Mailing Materials**	holder	letter	drawer	mail	product	hanging_folder	filing_cabinet	slot	stamp	monitor_arm
Topic 23	**Label maker**	label_maker	label	dymo	font	labeler	symbol	comb	border	brother	versatile

From here, we can further analyze sentiment around these topics keywords (e.g. search for adjectives or reviews star ratings associated). In business applications, this provides insight into which topics customers deem important, as well as how they feel about it. This enables targeted product development and customer experience improvements. This example contains a variety of products, but a separate topic model into each product may reveal aspects that customers care about. For example, this analysis already started to reveal important aspects of calculators (topic 21) such as display, easy to press buttons, battery, weight. Sellers then need to make sure to highlight these features in their product descriptions or improve

visualizing and interpreting topics

Data Science Naturallanguageprocessing Machine Learning NLP Topic Modeling