IN-DEPTH ANALYSIS

Evaluate Topic Models: Latent Dirichlet Allocation (LDA)

A step-by-step guide to building interpretable topic models

Shashank Kapadia Aug 19, 2019 · 12 min read

Preface: This article aims to provide consolidated information on the underlying topic and is not to be considered as the original work. The information and the code are repurposed through several online articles, research papers, books, and open-source code

In the previous article, I introduced the concept of topic modeling and walked through the code for developing your first topic model using Latent Dirichlet Allocation (LDA) method in the python using *Gensim* implementation.

Pursuing on that understanding, in this article, we'll go a few steps deeper by outlining the framework to quantitatively evaluate topic models through the measure of topic coherence and share the code template in python using *Gensim* implementation to allow for end-to-end model development.

Why evaluate topic models?

We know probabilistic topic models, such as LDA, are popular tools for text analysis, providing both a predictive and latent topic representation of the corpus. However, there is a longstanding assumption that the latent space discovered by these models is generally meaningful and useful, and that evaluating such assumptions is challenging due to its unsupervised training process. Besides, there is a no-gold standard list of topics to compare against every corpus.

Nevertheless, it is equally important to identify if a trained model is objectively good or bad, as well have an ability to compare different models/methods. To do so, one would require an objective measure for the quality. Traditionally, and still for many practical applications, to evaluate if "the correct thing" has been learned about the corpus, an implicit knowledge and "eyeballing" approaches are used. Ideally, we'd like to capture this information in a single metric that can be maximized, and compared.

Let's take a look at roughly what approaches are commonly used for the evaluation:

Eye Balling Models

- Top N words
- Topics / Documents

Intrinsic Evaluation Metrics

- Capturing model semantics
- Topics interpretability

Human Judgements

- What is a topic

Extrinsic Evaluation Metrics/Evaluation at task

- Is model good at performing predefined tasks, such as classification

this article, we'll explore more about topic coherence, an intrinsic evaluation metric, and how you can use it to quantitatively justify the model selection.

What is Topic Coherence?

Before we understand topic coherence, let's briefly look at the perplexity measure. Perplexity as well is one of the intrinsic evaluation metric, and is widely used for language model evaluation. It captures how surprised a model is of new data it has not seen before, and is measured as the normalized log-likelihood of a held-out test set.

Focussing on the log-likelihood part, you can think of the perplexity metric as measuring how probable some new unseen data is given the model that was learned earlier. That is to say, how well does the model represent or reproduce the statistics of the held-out data.

However, recent studies have shown that predictive likelihood (or equivalently, perplexity) and human judgment are often not correlated, and even sometimes slightly anti-correlated.

> *Optimizing for perplexity may not yield human interpretable topics*

This limitation of perplexity measure served as a motivation for more work trying to model the human judgment, and thus *Topic Coherence*.

The concept of topic coherence combines a number of measures into a framework to evaluate the coherence between topics inferred by a model. But before that…

What is topic coherence?

Topic Coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. These measurements help distinguish between topics that are semantically interpretable topics and topics that are artifacts of statistical inference. But …

What is coherence?

example of a coherent fact set is “the game is a team sport”, “the game is played with a ball”, “the game demands great physical efforts”

Coherence Measures

Let’s take quick look at different coherence measures, and how they are calculated:

1. ***C_v*** measure is based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity
2. ***C_p*** is based on a sliding window, one-preceding segmentation of the top words and the confirmation measure of Fitelson’s coherence
3. ***C_uci*** measure is based on a sliding window and the pointwise mutual information (PMI) of all word pairs of the given top words
4. ***C_umass*** is based on document cooccurrence counts, a one-preceding segmentation and a logarithmic conditional probability as confirmation measure
5. ***C_npmi*** is an enhanced version of the C_uci coherence using the normalized pointwise mutual information (NPMI)
6. ***C_a*** is baseed on a context window, a pairwise comparison of the top words and an indirect confirmation measure that uses normalized pointwise mutual information (NPMI) and the cosine similarity

There is, of course, a lot more to the concept of topic model evaluation, and the coherence measure. However, keeping in mind the length, and purpose of this article, let’s apply these concepts into developing a model that is at least better than with the default parameters. Also, we’ll be re-purposing already available online pieces of code to support this exercise instead of re-inventing the wheel.

Model Implementation

1. Loading data
2. Data Cleaning
3. Phrase Modeling: Bi-grams and Tri-grams
4. Data transformation: Corpus and Dictionary
5. Base Model
6. Hyperparameter Tuning
7. Final Model
8. Visualize Results

Loading Data

For this tutorial, we'll use the dataset of papers published in NIPS conference. The NIPS conference (Neural Information Processing Systems) is one of the most prestigious yearly events in the machine learning community. The CSV data file contains information on the different NIPS papers that were published from 1987 until 2016 (29 years!). These papers discuss a wide variety of topics in machine learning, from neural networks to optimization methods, and many more.

Let's start by looking at the content of the file

```
import os

os.chdir('..')

# Read data into papers
papers = pd.read_csv('./data/NIPS Papers/papers.csv')

# Print head
papers.head()
```

	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\n\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

Data Cleaning

Since the goal of this analysis is to perform topic modeling, we will solely focus on the text data from each paper, and drop other metadata columns

```
# Remove the columns
papers = papers.drop(columns=['id', 'title', 'abstract',
                              'event_type', 'pdf_name', 'year'],
axis=1)

# sample only 100 papers
papers = papers.sample(100)

# Print out the first rows of papers
papers.head()
```

Remove punctuation/lower casing

Next, let's perform a simple preprocessing on the content of paper_text column to make them more amenable for analysis, and reliable results. To do that, we'll use a regular expression to remove any punctuation, and then lowercase the text

```
# Remove punctuation
papers['paper_text_processed'] = papers['paper_text'].map(lambda x:
re.sub('[,\.!?]', '', x))

# Convert the titles to lowercase
papers['paper_text_processed'] =
papers['paper_text_processed'].map(lambda x: x.lower())

# Print out the first rows of papers
papers['paper_text_processed'].head()
```

```
5263    the human kernel\nandrew gordon wilson\ncmu\n\...
3654    nonstandard interpretations of probabilistic\n...
1440    388\n\nsmith and miller\n\nbayesian inference ...
4311    message passing inference with chemical reacti...
1489    denoising and untangling graphs using\ndegree ...
Name: paper_text_processed, dtype: object
```

Tokenize words and further clean-up text

Let's tokenize each sentence into a list of words, removing punctuations and unnecessary characters altogether.

```
import gensim
from gensim.utils import simple_preprocess

def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence),
deacc=True))  # deacc=True removes punctuations

data = papers.paper_text_processed.values.tolist()
data_words = list(sent_to_words(data))

print(data_words[:1][0][:30])
```

```
[u'the', u'human', u'kernel', u'andrew', u'gordon', u'wilson', u'cmu', u'christoph', u'dann', u'cmu', u'christopher', u'lucas', u'university', u'of', u'edinburgh', u'eric', u'xing', u'cmu', u'ab
stract', u'bayesian', u'nonparametric', u'models', u'such', u'as', u'gaussian', u'processes', u'provide', u'compelling', u'framework', u'for', u'automatic', u'statistical', u'modelling', u'thes
e', u'models', u'have', u'high', u'degree', u'of', u'flexibility', u'and', u'automatically', u'calibrated', u'complexity', u'however', u'automating', u'human', u'expertise', u'remains', u'elusiv
e', u'for', u'example', u'gaussian', u'processes', u'with', u'standard', u'kernels', u'struggle', u'on', u'function', u'extrapolation', u'problems', u'that', u'are', u'trivial', u'for', u'human',
u'learners', u'in', u'this', u'paper', u'we', u'create', u'function', u'extrapolation', u'problems', u'and', u'acquire', u'human', u'responses', u'and', u'then', u'design', u'kernel', u'learnin
g', u'framework', u'to', u'reverse', u'engineer', u'the', u'inductive', u'biases', u'of', u'human', u'learners', u'across', u'set', u'of', u'behavioral', u'experiments', u'we', u'use', u'the',
u'learned', u'kernels', u'to', u'gain', u'psychological', u'insights', u'and', u'to', u'extrapolate', u'in', u'humanlike', u'ways', u'that', u'go', u'beyond', u'traditional', u'stationary', u'an
d', u'polynomial', u'kernels', u'finally', u'we', u'investigate', u'occams', u'razor', u'in', u'human', u'and', u'gaussian', u'process', u'based', u'function', u'learning', u'introduction', u'tru
ly', u'intelligent', u'systems', u'can', u'learn', u'and', u'make', u'decisions', u'without', u'human', u'intervention', u'therefore', u'it', u'is', u'not', u'surprising', u'that', u'early', u'ma
chine', u'learning', u'efforts', u'such', u'as', u'the', u'perceptron', u'have', u'been', u'neurally', u'inspired', u'in', u'recent', u'years', u'probabilistic', u'modelling', u'has', u'become',
u'cornerstone', u'of', u'machine', u'learning', u'approaches', u'with', u'applications', u'in', u'neural', u'processing', u'and', u'human', u'learning', u'from', u'probabilistic', u'perspective',
u'the', u'ability', u'for', u'model', u'to', u'automatically', u'discover', u'patterns', u'and', u'perform', u'extrapolation', u'is', u'determined', u'by', u'its', u'support', u'which', u'solutio
ns', u'are', u'priori', u'possible', u'and', u'inductive', u'biases', u'which', u'solutions', u'are', u'priori', u'likely', u'ideally', u'we', u'want', u'model', u'to', u'be', u'able', u'to', u'r
epresent', u'many', u'possible', u'solutions', u'to', u'given', u'problem', u'with', u'inductive', u'biases', u'which', u'can', u'extract', u'intricate', u'structure', u'from', u'limited', u'dat
```

Bigrams are two words frequently occurring together in the document. Trigrams are 3 words frequently occurring. Some examples in our example are: 'back_bumper', 'oil_leakage', 'maryland_college_park' etc.

Gensim's Phrases model can build and implement the bigrams, trigrams, quadgrams and more. The two important arguments to Phrases are min_count and threshold.

> *The higher the values of these param, the harder it is for words to be combined.*

```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5,
threshold=100) # higher threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

Remove Stopwords, Make Bigrams and Lemmatize

The phrase models are ready. Let's define the functions to remove the stopwords, make trigrams and lemmatization and call them sequentially.

```
# NLTK Stop words
# import nltk
# nltk.download('stopwords')
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])

# Define functions for stopwords, bigrams, trigrams and lemmatization

def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not
in stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]
```

```
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB',
'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_
in allowed_postags])
    return texts_out
```

Let's call the functions in order.

```
import spacy

# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for
efficiency)
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=
['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```


Data Transformation: Corpus and Dictionary

The two main inputs to the LDA topic model are the dictionary(id2word) and the corpus. Let's create them.

```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```


Gensim creates a unique id for each word in the document. The produced corpus shown above is a mapping of (word_id, word_frequency).

For example, (0, 7) above implies, word id 0 occurs seven times in the first document. Likewise, word id 1 occurs thrice and so on

Base Model

We have everything required to train the base LDA model. In addition to the corpus and dictionary, you need to provide the number of topics as well. Apart from that, alpha and eta are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior (we'll use default for the base model).

> ***chunksize*** *controls how many documents are processed at a time in the training algorithm. Increasing chunksize will speed up training, at least as long as the chunk of documents easily fit into memory.*
>
> ***passes*** *controls how often we train the model on the entire corpus (set to 10). Another word for passes might be "epochs". iterations is somewhat technical, but essentially it controls*

```
# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=10,
                                       random_state=100,
                                       chunksize=100,
                                       passes=10,
                                       per_word_topics=True)
```

View the topics in LDA model

The above LDA model is built with 10 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic.

You can see the keywords for each topic and the weightage(importance) of each keyword using *lda_model.print_topics()*

```
from pprint import pprint

# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

Compute Model Perplexity and Coherence Score

Let's calculate the baseline coherence score

```
coherence_model_lda = CoherenceModel(model=lda_model,
texts=data_lemmatized, dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()

print('\nCoherence Score: ', coherence_lda)
```

Coherence Score: 0.301

Hyperparameter Tuning

First, let's differentiate between model hyperparameters and model parameters :

> ***Model hyperparameters*** *can be thought of as settings for a machine learning algorithm that are tuned by the data scientist before training. Examples would be the number of trees in the random forest, or in our case, number of topics K*
>
> ***Model parameters*** *can be thought of as what the model learns during training, such as the weights for each word in a given topic*

Now that we have the baseline coherence score for the default LDA model, let's perform a series of sensitivity tests to help determine the following model hyperparameters:

1. Number of Topics (K)
2. Dirichlet hyperparameter alpha: Document-Topic Density
3. Dirichlet hyperparameter beta: Word-Topic Density

We'll perform these tests in sequence, one parameter at a time by keeping others constant and run them over the two different validation corpus sets. We'll use ***C_v*** as our choice of metric for performance comparison

```
# supporting function
def compute_coherence_values(corpus, dictionary, k, a, b):

    lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                           id2word=dictionary,
                                           num_topics=k,
```

```
                                                alpha=a,
                                                eta=b)

    coherence_model_lda = CoherenceModel(model=lda_model,
texts=data_lemmatized, dictionary=id2word, coherence='c_v')

    return coherence_model_lda.get_coherence()
```

Let's call the function, and iterate it over the range of topics, alpha, and beta parameter values

```
import numpy as np
import tqdm

grid = {}
grid['Validation_Set'] = {}

# Topics range
min_topics = 2
max_topics = 11
step_size = 1
topics_range = range(min_topics, max_topics, step_size)

# Alpha parameter
alpha = list(np.arange(0.01, 1, 0.3))
alpha.append('symmetric')
alpha.append('asymmetric')

# Beta parameter
beta = list(np.arange(0.01, 1, 0.3))
beta.append('symmetric')

# Validation sets
num_of_docs = len(corpus)
corpus_sets = [# gensim.utils.ClippedCorpus(corpus,
num_of_docs*0.25),
               # gensim.utils.ClippedCorpus(corpus, num_of_docs*0.5),
               gensim.utils.ClippedCorpus(corpus, num_of_docs*0.75),
               corpus]

corpus_title = ['75% Corpus', '100% Corpus']

model_results = {'Validation_Set': [],
                 'Topics': [],
                 'Alpha': [],
```

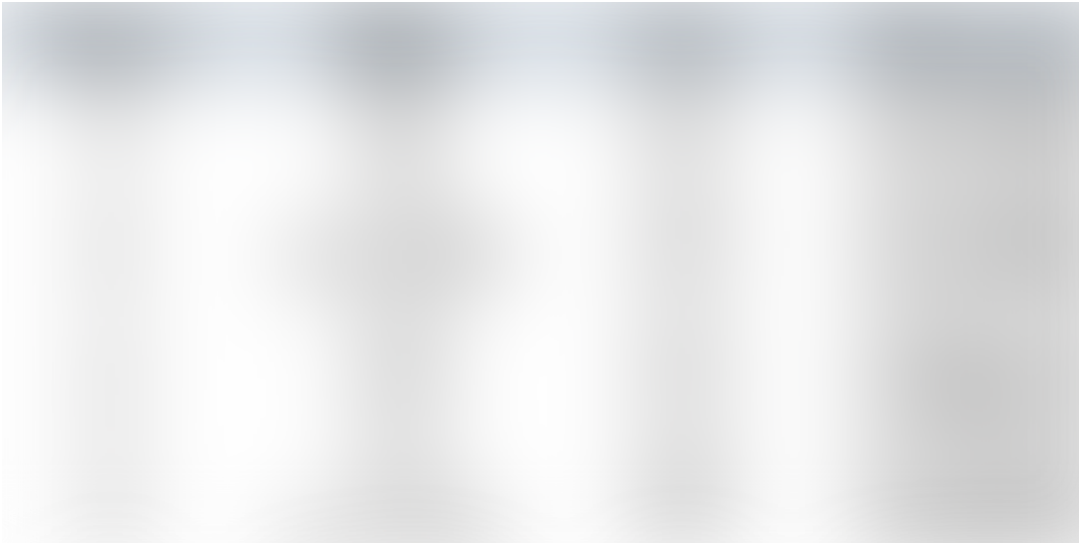
```
# Can take a long time to run
if 1 == 1:
    pbar = tqdm.tqdm(total=540)

    # iterate through validation corpuses
    for i in range(len(corpus_sets)):
        # iterate through number of topics
        for k in topics_range:
            # iterate through alpha values
            for a in alpha:
                # iterare through beta values
                for b in beta:
                    # get the coherence score for the given
parameters
                    cv =
compute_coherence_values(corpus=corpus_sets[i], dictionary=id2word,
                                                  k=k, a=a, b=b)
                    # Save the model results

model_results['Validation_Set'].append(corpus_title[i])
                    model_results['Topics'].append(k)
                    model_results['Alpha'].append(a)
                    model_results['Beta'].append(b)
                    model_results['Coherence'].append(cv)

                    pbar.update(1)
    pd.DataFrame(model_results).to_csv('lda_tuning_results.csv',
index=False)
    pbar.close()
```

Investigate Results

Let's start by determining the optimal number of topics. The chart below outlines the coherence score, C_v, for the number of topics across two validation sets, and a fixed alpha = 0.01 and beta = 0.1

With the coherence score seems to keep increasing with the number of topics, it may make better sense to pick the model that gave the highest CV before flattening out or a major drop. In this case, we picked K=8

Next, we want to select the optimal alpha and beta parameters. While there are other sophisticated approaches to tackle the selection process, for this tutorial, we choose the values that yielded maximum C_v score for K=8



> *alpha=0.01*
>
> *beta=0.9*
>
> *K=8*

That yields approx. 17% improvement over the baseline score

Final Model

Let's train the final model using the above selected parameters

```
num_topics=8,
random_state=100,
chunksize=100,
passes=10,
alpha=0.01,
eta=0.9)
```

Visualize Topics

```
import pyLDAvis.gensim
import pickle
import pyLDAvis

# Visualize the topics
pyLDAvis.enable_notebook()

LDAvis_prepared = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)

LDAvis_prepared
```

reviewed existing methods and scratched the surface of topic coherence, along with the available coherence measures. Then we built a default LDA model using Gensim implementation to establish the baseline coherence score and reviewed practical ways to optimize the LDA hyperparameters.

Hopefully, this article has managed to shed light on the underlying topic evaluation strategies, and intuitions behind it.

References:

1. http://qpleple.com/perplexity-to-evaluate-topic-models/
2. https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020
3. https://papers.nips.cc/paper/3700-reading-tea-leaves-how-humans-interpret-topic-models.pdf
4. https://github.com/mattilyra/pydataberlin-2017/blob/master/notebook/EvaluatingUnsupervisedModels.ipynb
5. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
6. http://svn.aksw.org/papers/2015/WSDM_Topic_Evaluation/public.pdf
7. http://palmetto.aksw.org/palmetto-webapp/

Thanks for reading. *If you have any feedback, please feel to reach out by commenting on this post, messaging me on LinkedIn, or shooting me an email (shmkapadia[at]gmail.com)*

If you enjoyed this article, visit my other articles

Topic Modeling in Python: Latent Dirichlet Allocation (LDA)

Building Blocks: Text Pre-Processing

This article is the second of more to come articles on Natural Language Processing. The purpose of this series of...

towardsdatascience.com

Introduction to Language Models: N-Gram

This article is the third of more to come articles on Natural Language Processing. The purpose of this series of...

towardsdatascience.com