

LDA in Python – How to grid search best topic models?

by [Selva Prabhakaran](https://www.machinelearningplus.com/author/selva86/) (https://www.machinelearningplus.com/author/selva86/). | Posted on April 4, 2018 (https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/)

Python's Scikit Learn provides a convenient interface for topic modeling using algorithms like Latent Dirichlet allocation(LDA), LSI and Non-Negative Matrix Factorization. In this tutorial, you will learn how to build the best possible LDA topic model and explore how to showcase the outputs as meaningful results.

[container] [columnize] [1. Introduction](#)

Contents

- [2. Load the packages](#)
- [3. Import Newsgroups Text Data](#)
- [4. Remove emails and newline characters](#)
- [5. Tokenize and Clean-up using gensim's simple preprocess\(\)](#)
- [6. Lemmatization](#)
- [7. Create the Document-Word matrix](#)
- [8. Check the Sparsity](#)
- [9. Build LDA model with sklearn](#)
- [10. Diagnose model performance with perplexity and log-likelihood](#)
- [11. How to GridSearch the best LDA model?](#)
- [12. How to see the best topic model and its parameters?](#)
- [13. Compare LDA Model Performance Scores](#)
- [14. How to see the dominant topic in each document?](#)
- [15. Review topics distribution across documents](#)
- [16. How to visualize the LDA model with pyLDAvis?](#)
- [17. How to see the Topic's keywords?](#)
- [18. Get the top 15 keywords each topic](#)
- [19. How to predict the topics for a new piece of text?](#)
- [20. How to cluster documents that share similar topics and plot?](#)
- [21. How to get similar documents for any given piece of text?](#)
- [22. Conclusion](#)

[/columnize]

Search ...

Search

Recent Posts

[data.table in R – The Complete Beginners Guide](#)

(https://www.machinelearningplus.com/data-manipulation/datatable-in-r-complete-guide/)

[Augmented Dickey Fuller Test \(ADF Test\) – Must Read Guide](#)

(https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/)

[KPSS Test for Stationarity](#)

(https://www.machinelearningplus.com/time-series/kpss-test-for-stationarity/)

[101 R data.table Exercises](#)

(https://www.machinelearningplus.com/data-manipulation/101-r-data-table-exercises/)

[P-Value – Understanding from Scratch](#)

(https://www.machinelearningplus.com/statistics/p-value/)

[101 Python datatable Exercises \(pydatatable\)](#)

(https://www.machinelearningplus.com/data-manipulation/101-python-datatable-exercises-pydatatable/)

[Vector Autoregression \(VAR\) –](#)

[Comprehensive Guide with Examples in Python](#)

(https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/)

[Mahalanobis Distance – Understanding the math with examples \(python\)](#)

(https://www.machinelearningplus.com/statistics/mahalanobis-distance/)

[datetime in Python – Simplified Guide with Clear Examples](#)

(https://www.machinelearningplus.com/python/datetime-examples/)

[Principal Component Analysis \(PCA\) – Better Explained](#)

(https://www.machinelearningplus.com/machine-learning/principal-components-analysis-pca-better-explained/)

[Python Logging – Simplest Guide with Full Code and Examples](#)

(https://www.machinelearningplus.com/python/python-logging-guide/)



How to build topic models with python sklearn. Photo by Sebastien Gabriel.

1. Introduction

In the last tutorial you saw [how to build topics models with LDA using gensim](https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/) (<https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/>). In this tutorial, however, I am going to use python's the most popular machine learning library – [scikit learn](http://scikit-learn.org/stable/) (<http://scikit-learn.org/stable/>).

With scikit learn, you have an entirely different interface and with grid search and vectorizers, you have a lot of options to explore in order to find the optimal model and to present the results.

In this tutorial, you will learn:

1. How to clean and process text data?
2. How to prepare the text documents to build topic models with scikit learn?
3. How to build a basic topic model using LDA and understand the params?
4. How to extract the topic's keywords?
5. How to gridsearch and tune for optimal model?
6. How to get the dominant topics in each document?
7. Review and visualize the topic keywords distribution
8. How to predict the topics for a new piece of text?
9. Cluster the documents based on topic distribution
10. How to get most similar documents based on topics discussed?

A lot of exciting stuff ahead. Let's roll!

2. Load the packages

The core package used in this tutorial is scikit-learn (sklearn).

Regular expressions `re`, `gensim` and `spacy` are used to process texts. `pyLDAvis` and `matplotlib` for visualization and `numpy` and `pandas` for manipulating and viewing data in tabular format.

Let's import them.

[Matplotlib Histogram – How to Visualize Distributions in Python](https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/)
(<https://www.machinelearningplus.com/plots/matplotlib-histogram-python-examples/>).

[ARIMA Model – Complete Guide to Time Series Forecasting in Python](https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/)
(<https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/>).

[Time Series Analysis in Python – A Comprehensive Guide with Examples](https://www.machinelearningplus.com/time-series/time-series-analysis-python/)
(<https://www.machinelearningplus.com/time-series/time-series-analysis-python/>).

[Matplotlib Tutorial – A Complete Guide to Python Plot w/ Examples](https://www.machinelearningplus.com/plots/matplotlib-tutorial-complete-guide-python-plot-examples/)
(<https://www.machinelearningplus.com/plots/matplotlib-tutorial-complete-guide-python-plot-examples/>).

[Topic modeling visualization – How to present the results of LDA models?](https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/)
(<https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/>).

[Top 50 matplotlib Visualizations – The Master Plots \(with full python code\)](https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/)
(<https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/>).

[List Comprehensions in Python – My Simplified Guide](https://www.machinelearningplus.com/python/list-comprehensions-in-python/)
(<https://www.machinelearningplus.com/python/list-comprehensions-in-python/>).

[Python @Property Explained – How to Use and When? \(Full Examples\)](https://www.machinelearningplus.com/python/python-property/)
(<https://www.machinelearningplus.com/python/python-property/>).

[How Naive Bayes Algorithm Works? \(with example and full code\)](https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/)
(<https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/>).

Tags

[Bigrams](https://www.machinelearningplus.com/tag/bigrams/) (<https://www.machinelearningplus.com/tag/bigrams/>).

[Classification](https://www.machinelearningplus.com/tag/classification/)

(<https://www.machinelearningplus.com/tag/classification/>).

[Corpus](https://www.machinelearningplus.com/tag/corpus/) (<https://www.machinelearningplus.com/tag/corpus/>).

[Cosine Similarity](https://www.machinelearningplus.com/tag/cosine-similarity/)

(<https://www.machinelearningplus.com/tag/cosine-similarity/>).

[data.table](https://www.machinelearningplus.com/tag/data-table/)

(<https://www.machinelearningplus.com/tag/data-table/>).

[Data Manipulation](https://www.machinelearningplus.com/tag/data-manipulation/)

(<https://www.machinelearningplus.com/tag/data-manipulation/>).

[Debugging](https://www.machinelearningplus.com/tag/debugging/)

(<https://www.machinelearningplus.com/tag/debugging/>).

[Doc2Vec](https://www.machinelearningplus.com/tag/doc2vec/)
(<https://www.machinelearningplus.com/tag/doc2vec/>).

Feedback

```
# Run in terminal or command prompt
# python3 -m spacy download en

import numpy as np
import pandas as pd
import re, nltk, spacy, gensim

# Sklearn
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from pprint import pprint

# Plotting tools
import pyLDAvis
import pyLDAvis.sklearn
import matplotlib.pyplot as plt
%matplotlib inline
```

3. Import Newsgroups Text Data

I will be using the 20-Newsgroups dataset for this. This version of the dataset contains about 11k newsgroups posts from 20 different topics. This is available as [newsgroups.json](https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json) (<https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json>).

Since it is in a json format with a consistent structure, I am using `pandas.read_json()` and the resulting dataset has 3 columns as shown.

```
# Import Dataset
df = pd.read_json('https://raw.githubusercontent.com/selva86/datasets/master/newsgroups.json')
print(df.target_names.unique())
```

```
['rec.autos' 'comp.sys.mac.hardware' 'rec.motorcycles' 'misc.forsale'
 'comp.os.ms-windows.misc' 'alt.atheism' 'comp.graphics'
 'rec.sport.baseball' 'rec.sport.hockey' 'sci.electronics' 'sci.space'
 'talk.politics.misc' 'sci.med' 'talk.politics.mideast'
 'soc.religion.christian' 'comp.windows.x' 'comp.sys.ibm.pc.hardware'
 'talk.politics.guns' 'talk.religion.misc' 'sci.crypt']
```

```
df.head(15)
```

Evaluation Metrics

(<https://www.machinelearningplus.com/tag/evaluation-metrics/>) [FastText](https://www.machinelearningplus.com/tag/fasttext/)

(<https://www.machinelearningplus.com/tag/fasttext/>).

Feature Selection

(<https://www.machinelearningplus.com/tag/feature-selection/>)

Gensim

(<https://www.machinelearningplus.com/tag/gensim>) [LDA](https://www.machinelearningplus.com/tag/lda/)

(<https://www.machinelearningplus.com/tag/lda/>).

Lemmatization

(<https://www.machinelearningplus.com/tag/lemmatization/>) [Linear Regression](https://www.machinelearningplus.com/tag/linear-regression/)

(<https://www.machinelearningplus.com/tag/linear-regression/>) [Logistic](https://www.machinelearningplus.com/tag/logistic/)

(<https://www.machinelearningplus.com/tag/logistic/>) [LSI](https://www.machinelearningplus.com/tag/lsi/)

(<https://www.machinelearningplus.com/tag/lsi/>) [Matplotlib](https://www.machinelearningplus.com/tag/matplotlib/)

(<https://www.machinelearningplus.com/tag/matplotlib/>) [Multiprocessing](https://www.machinelearningplus.com/tag/multiprocessing/)

(<https://www.machinelearningplus.com/tag/multiprocessing/>).

(<https://www.machinelearningplus.com/tag/multiprocessing/>).

NLP

(<https://www.machinelearningplus.com/tag/nlp/>)

[NLTK](https://www.machinelearningplus.com/tag/nltk/) (<https://www.machinelearningplus.com/tag/nltk/>).

Numpy

(<https://www.machinelearningplus.com/tag/numpy/>)

P-Value

(<https://www.machinelearningplus.com/tag/p-value/>) [Pandas](https://www.machinelearningplus.com/tag/pandas/)

(<https://www.machinelearningplus.com/tag/pandas/>) [Parallel Processing](https://www.machinelearningplus.com/tag/parallel-processing/)

(<https://www.machinelearningplus.com/tag/parallel-processing/>) [Phraser](https://www.machinelearningplus.com/tag/phraser/)

(<https://www.machinelearningplus.com/tag/phraser/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

(<https://www.machinelearningplus.com/tag/practice-exercise/>) [Practice Exercise](https://www.machinelearningplus.com/tag/practice-exercise/)

Python

(<https://www.machinelearningplus.com/tag/python/>)

R

(<https://www.machinelearningplus.com/tag/r/>)

(<https://www.machinelearningplus.com/tag/r/>) [Regex](https://www.machinelearningplus.com/tag/regex/)

Regression

(<https://www.machinelearningplus.com/tag/regression/>)

Residual Analysis

(<https://www.machinelearningplus.com/tag/residual-analysis/>)

(<https://www.machinelearningplus.com/tag/residual-analysis/>) [Scikit Learn](https://www.machinelearningplus.com/tag/scikit-learn/)

(<https://www.machinelearningplus.com/tag/scikit-learn/>) [Significance Tests](https://www.machinelearningplus.com/tag/significance-tests/)

(<https://www.machinelearningplus.com/tag/significance-tests/>) [Soft Cosine Similarity](https://www.machinelearningplus.com/tag/soft-cosine-similarity/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

(<https://www.machinelearningplus.com/tag/soft-cosine-similarity/>) [spaCy](https://www.machinelearningplus.com/tag/spacy/)

Stationarity

(<https://www.machinelearningplus.com/tag/stationarity/>)

Summarization

(<https://www.machinelearningplus.com/tag/summarization/>)

TaggedDocument

(<https://www.machinelearningplus.com/tag/taggeddocument/>)

(<https://www.machinelearningplus.com/tag/taggeddocument/>) [TextBlob](https://www.machinelearningplus.com/tag/textblob/)

(<https://www.machinelearningplus.com/tag/textblob/>) [TFIDF](https://www.machinelearningplus.com/tag/tfidf/)

(<https://www.machinelearningplus.com/tag/tfidf/>) [Time Series](https://www.machinelearningplus.com/tag/time-series/)

(<https://www.machinelearningplus.com/tag/time-series/>)

(<https://www.machinelearningplus.com/tag/time-series/>)

Feedback

	content	target	target_names
0	From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this?\nNntp-Posting-Host: rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '15 I was wondering if anyone out there could enlighten me on this car I saw 'the other day. It was a 2-door sports car, looked to be from the late 60s/ 'early 70s. It was called a Bricklin. The doors were really small. In 'addition, the front bumper was separate from the rest of the body. This is 'all I know. If anyone can tellme a model name, engine specs, years of 'production, where this car is made, history, or whatever info you have on 'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by 'your neighborhood Lerxst ---- '	7	rec.autos
1	From: guykuo@carson.u.washington.edu (Guy Kuo)\nSubject: SI Clock Poll - Fin...	4	comp.sys.mac.hardware
10	From: irwin@cmptrc.lonestar.org (Irwin Arnstein)\nSubject: Re: Recommendation...	8	rec.motorcycles
100	From: tchen@magnus.acs.ohio-state.edu (Tsung-Kun Chen)\nSubject: ** Software...	6	misc.forsale
1000	From: dabl2@nml.nih.gov (Don A.B. Lindbergh)\nSubject: Diamond SS24X, Win 3....	2	comp.os.ms-windows.misc
10000	From: a207706@moe.dseg.ti.com (Robert Loper)\nSubject: Re: SHO and SC\nNntp-Posting-Host: rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '15 I was wondering if anyone out there could enlighten me on this car I saw 'the other day. It was a 2-door sports car, looked to be from the late 60s/ 'early 70s. It was called a Bricklin. The doors were really small. In 'addition, the front bumper was separate from the rest of the body. This is 'all I know. If anyone can tellme a model name, engine specs, years of 'production, where this car is made, history, or whatever info you have on 'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by 'your neighborhood Lerxst ---- '	7	rec.autos
10001	From: kimman@magnus.acs.ohio-state.edu (Kim Richard Man)\nSubject: SyQuest 4...	6	misc.forsale
10002	From: kwilson@casbah.acns.nwu.edu (Kirtley Wilson)\nSubject: Mirosoft Office...	2	comp.os.ms-windows.misc
10003	Subject: Re: Don't more innocents die without the death penalty?\nFrom: bobb...	0	alt.atheism
10004	From: livesey@solntze.wpd.sgi.com (Jon Livesey)\nSubject: Re: Genocide is Ca...	0	alt.atheism
10005	From: dls@aeg.dsto.gov.au (David Silver)\nSubject: Re: Fractal Generation of...	1	comp.graphics
10006	Subject: Re: Mike Francesa's 1993 Predictions\nFrom: gajarsky@pilot.njin.net...	9	rec.sport.baseball
10007	From: jet@netcom.Netcom.COM (J. Eric Townsend)\nSubject: Re: Insurance and L...	8	rec.motorcycles
10008	From: gld@cunibx.cc.columbia.edu (Gary L Dare)\nSubject: Re: ABC coverage\nNntp-Posting-Host: rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '15 I was wondering if anyone out there could enlighten me on this car I saw 'the other day. It was a 2-door sports car, looked to be from the late 60s/ 'early 70s. It was called a Bricklin. The doors were really small. In 'addition, the front bumper was separate from the rest of the body. This is 'all I know. If anyone can tellme a model name, engine specs, years of 'production, where this car is made, history, or whatever info you have on 'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by 'your neighborhood Lerxst ---- '	10	rec.sport.hockey
10009	From: sehari@iastate.edu (Babak Sehari)\nSubject: Re: How to the disks copy ...	12	sci.electronics

(https://machinelearningplus.sirv.com/WP-20NewsGroups-2018/04/i/n/s/input_texts.png).

Input – 20NewsGroups

4. Remove emails and newline characters

You can see many emails, newline characters and extra spaces in the text and it is quite distracting. Let's get rid of them using [regular expressions](https://www.machinelearningplus.com/python/python-regex-tutorial-examples/)

(<https://www.machinelearningplus.com/python/python-regex-tutorial-examples/>).

```
# Convert to list
data = df.content.values.tolist()

# Remove Emails
data = [re.sub('\S*\S*\s?', ' ', sent) for sent in data]

# Remove new line characters
data = [re.sub('\s+', ' ', sent) for sent in data]

# Remove distracting single quotes
data = [re.sub("'", "", sent) for sent in data]

pprint(data[:1])
```

```
['From: (wheres my thing) Subject: WHAT car is this? Nntp-Posting-Host: '
'rac3.wam.umd.edu Organization: University of Maryland, College Park Lines: '
'15 I was wondering if anyone out there could enlighten me on this car I saw '
'the other day. It was a 2-door sports car, looked to be from the late 60s/ '
'early 70s. It was called a Bricklin. The doors were really small. In '
'addition, the front bumper was separate from the rest of the body. This is '
'all I know. If anyone can tellme a model name, engine specs, years of '
'production, where this car is made, history, or whatever info you have on '
'this funky looking car, please e-mail. Thanks, - IL ---- brought to you by '
'your neighborhood Lerxst ---- '']
```

5. Tokenize and Clean-up using gensim's simple_preprocess()

The sentences look better now, but you want to tokenize each sentence into a list of words, removing punctuations and unnecessary characters altogether.

[series/\). Topic Modeling](https://www.machinelearningplus.com/tag/topic-modeling/)
(<https://www.machinelearningplus.com/tag/topic-modeling/>). [Visualization](https://www.machinelearningplus.com/tag/visualization/)
(<https://www.machinelearningplus.com/tag/visualization/>). [Word2Vec](https://www.machinelearningplus.com/tag/word2vec/) (<https://www.machinelearningplus.com/tag/word2vec/>).

Gensim's `simple_preprocess()` is great for this. Additionally I have set `deacc=True` to remove the punctuations.

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

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp', 'posting',
```

6. Lemmatization

Lemmatization is a process where we convert words to its root word.

For example: 'Studying' becomes 'Study', 'Meeting' becomes 'Meet', 'Better' and 'Best' becomes 'Good'.

The advantage of this is, we get to reduce the total number of unique words in the dictionary. As a result, the number of columns in the document-word matrix (created by CountVectorizer in the next step) will be denser with lesser columns.

You can expect better topics to be generated in the end.

```
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation \_\(https://spacy.io/api/annotation\)"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append(" ".join([token.lemma_ if token.lemma_ not in ['-PRON-'] else '' for token in doc]))
    return texts_out

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# Run in terminal: python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do Lemmatization keeping only Noun, Adj, Verb, Adverb
data_lemmatized = lemmatization(data_words, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:2])
```

```
['where s thing subject what car be nntp post host rac wam umd edu organization university mar
```

7. Create the Document-Word matrix

The LDA topic model algorithm requires a document word matrix as the main input.

You can create one using `CountVectorizer`. In the below code, I have configured the `CountVectorizer` to consider words that has occurred at least 10 times (`min_df`), remove built-in english stopwords, convert all words to lowercase, and a word can contain numbers and alphabets of at least length 3 in order to be qualified as a word.

So, to create the doc-word matrix, you need to first initialise the `CountVectorizer` class with the required configuration and then apply `fit_transform` to actually create the matrix.

Since most cells contain zeros, the result will be in the form of a sparse matrix to save memory.

If you want to materialize it in a 2D array format, call the `todense()` method of the sparse matrix like its done in the next step.

```
vectorizer = CountVectorizer(analyzer='word',
                             min_df=10,           # minimum reqd occurrences of a w
                             stop_words='english', # remove stop words
                             lowercase=True,       # convert all words to lowercase
                             token_pattern='[a-zA-Z0-9]{3,}', # num chars > 3
                             # max_features=50000,    # max number of uniq words
                             )

data_vectorized = vectorizer.fit_transform(data_lemmatized)
```

8. Check the Sparsicity

Sparsicity is nothing but the percentage of non-zero datapoints in the document-word matrix, that is `data_vectorized`.

Since most cells in this matrix will be zero, I am interested in knowing what percentage of cells contain non-zero values.

```
# Materialize the sparse data
data_dense = data_vectorized.todense()

# Compute Sparsicity = Percentage of Non-Zero cells
print("Sparsicity: ", ((data_dense > 0).sum())/data_dense.size)*100, "%")
```

```
Sparsicity:  0.775887569365 %
```

9. Build LDA model with sklearn

Everything is ready to build a Latent Dirichlet Allocation (LDA) model. Let's initialise one and call `fit_transform()` to build the LDA model.

For this example, I have set the `n_topics` as 20 based on prior knowledge about the dataset. Later we will find the optimal number using grid search.

```
# Build LDA Model
lda_model = LatentDirichletAllocation(n_topics=20,          # Number of topics
                                     max_iter=10,          # Max Learning iterations
                                     learning_method='online',
                                     random_state=100,      # Random state
                                     batch_size=128,        # n docs in each Learning iteration
                                     evaluate_every = -1,    # compute perplexity every n iterations
                                     n_jobs = -1,           # Use all available CPUs
                                     )

lda_output = lda_model.fit_transform(data_vectorized)

print(lda_model) # Model attributes
```

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                           evaluate_every=-1, learning_decay=0.7,
                           learning_method='online', learning_offset=10.0,
                           max_doc_update_iter=100, max_iter=10, mean_change_tol=0.001,
                           n_components=10, n_jobs=-1, n_topics=20, perp_tol=0.1,
                           random_state=100, topic_word_prior=None,
                           total_samples=1000000.0, verbose=0)
```

10. Diagnose model performance with perplexity and log-likelihood

A model with higher log-likelihood and lower perplexity ($\exp(-1 \cdot \text{log-likelihood per word})$) is considered to be good. Let's check for our model.

```
# Log Likelihood: Higher the better
print("Log Likelihood: ", lda_model.score(data_vectorized))

# Perplexity: Lower the better. Perplexity = exp(-1. * Log-Likelihood per word)
print("Perplexity: ", lda_model.perplexity(data_vectorized))

# See model parameters
pprint(lda_model.get_params())
```

```
Log Likelihood: -9965645.21463
Perplexity: 2061.88393838
{'batch_size': 128,
 'doc_topic_prior': None,
 'evaluate_every': -1,
 'learning_decay': 0.7,
 'learning_method': 'online',
 'learning_offset': 10.0,
 'max_doc_update_iter': 100,
 'max_iter': 10,
 'mean_change_tol': 0.001,
 'n_components': 10,
 'n_jobs': -1,
 'n_topics': 20,
 'perp_tol': 0.1,
 'random_state': 100,
 'topic_word_prior': None,
 'total_samples': 1000000.0,
 'verbose': 0}
```

On a different note, perplexity might not be the best measure to evaluate topic models because it doesn't consider the context and semantic associations between words. This can be captured using topic coherence measure, an example of this is described in the gensim tutorial I mentioned earlier.

11. How to GridSearch the best LDA model?

The most important tuning parameter for LDA models is `n_components` (number of topics). In addition, I am going to search `learning_decay` (which controls the learning rate) as well.

Besides these, other possible search params could be `learning_offset` (downweigh early iterations. Should be > 1) and `max_iter`. These could be worth experimenting if you have enough computing resources.

Be warned, the grid search constructs multiple LDA models for all possible combinations of param values in the `param_grid` dict. So, this process can consume a lot of time and resources.

```
# Define Search Param
search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}

# Init the Model
lda = LatentDirichletAllocation()

# Init Grid Search Class
model = GridSearchCV(lda, param_grid=search_params)

# Do the Grid Search
model.fit(data_vectorized)
```



```

GridSearchCV(cv=None, error_score='raise',
             estimator=LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=-1, learning_decay=0.7, learning_method=None,
             learning_offset=10.0, max_doc_update_iter=100, max_iter=10,
             mean_change_tol=0.001, n_components=10, n_jobs=1,
             n_topics=None, perp_tol=0.1, random_state=None,
             topic_word_prior=None, total_samples=1000000.0, verbose=0),
             fit_params=None, iid=True, n_jobs=1,
             param_grid={'n_topics': [10, 15, 20, 25, 30], 'learning_decay': [0.5, 0.7, 0.9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring=None, verbose=0)

```

12. How to see the best topic model and its parameters?

```

# Best Model
best_lda_model = model.best_estimator_

# Model Parameters
print("Best Model's Params: ", model.best_params_)

# Log Likelihood Score
print("Best Log Likelihood Score: ", model.best_score_)

# Perplexity
print("Model Perplexity: ", best_lda_model.perplexity(data_vectorized))

```

```

Best Model's Params: {'learning_decay': 0.9, 'n_topics': 10}
Best Log Likelihood Score: -3417650.82946
Model Perplexity: 2028.79038336

```

13. Compare LDA Model Performance Scores

Plotting the log-likelihood scores against num_topics, clearly shows number of topics = 10 has better scores. And learning_decay of 0.7 outperforms both 0.5 and 0.9.

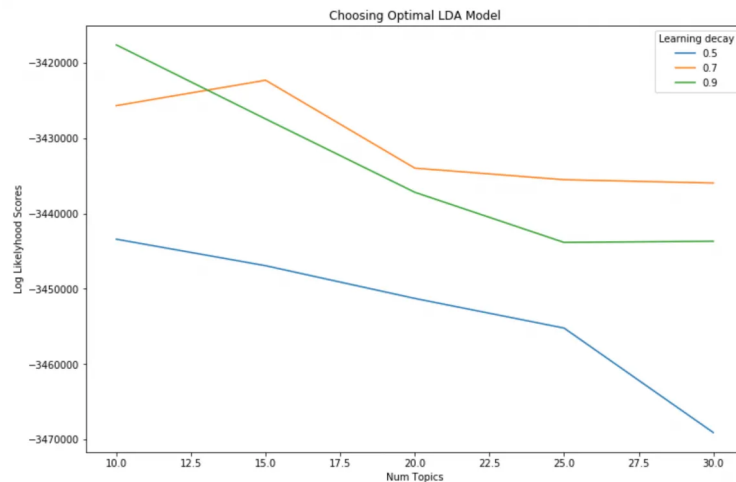
This makes me think, even though we know that the dataset has 20 distinct topics to start with, some topics could share common keywords. For example, 'alt.atheism' and 'soc.religion.christian' can have a lot of common words. Same with 'rec.motorcycles' and 'rec.autos', 'comp.sys.ibm.pc.hardware' and 'comp.sys.mac.hardware', you get the idea.

To tune this even further, you can do a finer grid search for number of topics between 10 and 15. But I am going to skip that for now.

So the bottom line is, a lower optimal number of distinct topics (even 10 topics) may be reasonable for this dataset. I don't know that yet. But LDA says so. Let's see.

```
# Get Log Likelyhoods from Grid Search Output
n_topics = [10, 15, 20, 25, 30]
log_likelihoods_5 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gs
log_likelihoods_7 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gs
log_likelihoods_9 = [round(gscore.mean_validation_score) for gscore in model.grid_scores_ if gs

# Show graph
plt.figure(figsize=(12, 8))
plt.plot(n_topics, log_likelihoods_5, label='0.5')
plt.plot(n_topics, log_likelihoods_7, label='0.7')
plt.plot(n_topics, log_likelihoods_9, label='0.9')
plt.title("Choosing Optimal LDA Model")
plt.xlabel("Num Topics")
plt.ylabel("Log Likelihood Scores")
plt.legend(title='Learning decay', loc='best')
plt.show()
```



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/g/r/s/grid-search-topic-models.png).

Grid Search Topic Models

14. How to see the dominant topic in each document?

To classify a document as belonging to a particular topic, a logical approach is to see which topic has the highest contribution to that document and assign it.

In the table below, I've greened out all major topics in a document and assigned the most dominant topic in its own column.

```

# Create Document - Topic Matrix
lda_output = best_lda_model.transform(data_vectorized)

# column names
topicnames = ["Topic" + str(i) for i in range(best_lda_model.n_topics)]

# index names
docnames = ["Doc" + str(i) for i in range(len(data))]

# Make the pandas dataframe
df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns=topicnames, index=docnames)

# Get dominant topic for each document
dominant_topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['dominant_topic'] = dominant_topic

# Styling
def color_green(val):
    color = 'green' if val > .1 else 'black'
    return 'color: {col}'.format(col=color)

def make_bold(val):
    weight = 700 if val > .1 else 400
    return 'font-weight: {weight}'.format(weight=weight)

# Apply Style
df_document_topics = df_document_topic.head(15).style.applymap(color_green).applymap(make_bold)
df_document_topics

```

	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	dominant_topic
Doc0	0	0	0	0	0	0.14	0	0	0	0.84	9
Doc1	0	0.05	0	0	0.05	0.24	0	0.65	0	0	7
Doc2	0	0	0	0	0	0.08	0.2	0	0	0.71	9
Doc3	0	0.55	0	0	0	0.44	0	0	0	0	1
Doc4	0.16	0.29	0	0	0	0.53	0	0	0	0	5
Doc5	0	0	0.05	0	0	0	0	0.12	0	0.83	9
Doc6	0	0	0	0	0	0.88	0.1	0	0	0	5
Doc7	0	0	0	0	0	0.99	0	0	0	0	5
Doc8	0	0	0.08	0.67	0	0	0	0	0.24	0	3
Doc9	0	0	0.74	0	0	0.14	0	0	0.11	0	2
Doc10	0	0	0	0	0.41	0.16	0	0.06	0	0.36	4
Doc11	0	0	0	0	0	0	0	0.97	0	0	7
Doc12	0	0	0	0.44	0	0.04	0	0.27	0	0.24	3
Doc13	0.14	0	0	0	0	0.07	0.57	0.08	0	0.13	6
Doc14	0	0	0	0	0.78	0.22	0	0	0	0	4

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/d/o/s/document_topic_weights.png).

Document Topic Weights: df_document_topics

15. Review topics distribution across documents

```

df_topic_distribution = df_document_topic['dominant_topic'].value_counts().reset_index(name="Num Documents")
df_topic_distribution.columns = ['Topic Num', 'Num Documents']
df_topic_distribution

```

Topic Num	Num Documents
0	5
1	2
2	4
3	9
4	1
5	7
6	3
7	8
8	0
9	6

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/t/o/n/topic_document_c

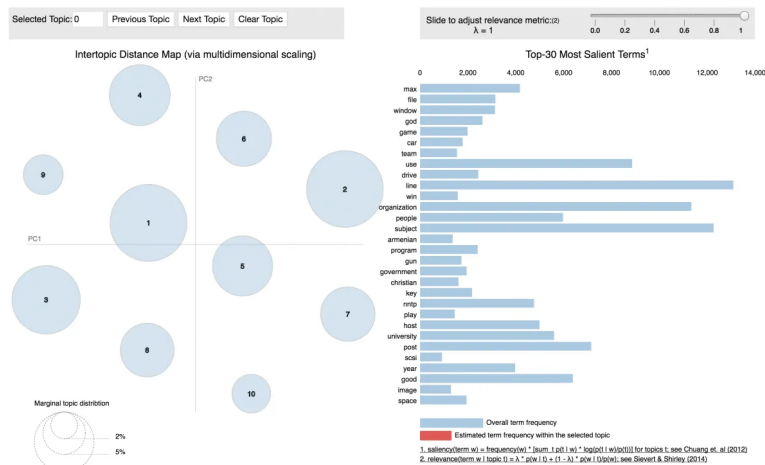
Topic Document Distribution:
df_topic_distribution

16. How to visualize the LDA model with pyLDAvis?

The pyLDAvis offers the best visualization to view the topics-keywords distribution.

A good topic model will have non-overlapping, fairly big sized blobs for each topic. This seems to be the case here. So, we are good.

```
pyLDAvis.enable_notebook()
panel = pyLDAvis.sklearn.prepare(best_lda_model, data_vectorized, vectorizer, mds='tsne')
panel
```



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/T/o/s/Topic-Distribution-using-pyLDAvis.png).

Visualize Topic Distribution using pyLDAvis

17. How to see the Topic's keywords?

The weights of each keyword in each topic is contained in `lda_model.components_` as a 2d array. The names of the keywords itself can be obtained from `vectorizer` object using `get_feature_names()`.

Let's use this info to construct a weight matrix for all keywords in each topic.

```
# Topic-Keyword Matrix
df_topic_keywords = pd.DataFrame(best_lda_model.components_)

# Assign Column and Index
df_topic_keywords.columns = vectorizer.get_feature_names()
df_topic_keywords.index = topicnames

# View
df_topic_keywords.head()
```

	aaa	aaron	abandon	abbreviation	abc	abide	ability	able	abolish	abomination	...	zion	zionism	zionist	zip	zitslein	zone	zoology
Topic0	9.575406	0.359981	0.100651	0.101689	0.117518	0.103411	3.453262	0.191085	0.107853	0.101680	...	0.100883	0.100633	0.100840	0.104864	0.100637	0.138300	0.100731
Topic1	0.100890	0.105746	0.131275	1.755703	0.101024	0.100685	19.461132	120.830022	0.104326	0.100753	...	0.100750	0.100967	0.100638	153.218332	0.100576	0.101657	0.100688
Topic2	0.109275	18.638578	31.982522	0.109649	0.110307	0.109873	85.985221	163.572888	23.501138	2.393444	...	0.101008	0.100948	0.102842	0.299180	0.100691	0.303542	0.100890
Topic3	0.103078	0.179178	0.101424	0.106328	0.101892	0.134686	5.270624	65.015866	0.100779	0.102793	...	1.246157	0.100726	0.102040	0.103102	29.849950	0.249888	82.317755
Topic4	0.100805	0.101627	1.919950	0.409059	6.007337	75.857222	87.171712	198.095427	27.465725	0.210156	...	0.256475	0.121024	0.143113	16.549160	0.115601	2.433356	0.101339

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/t/o/s/topic_word_weights.png).

Topic Word Weights: df_topic_keywords

18. Get the top 15 keywords each topic

From the above output, I want to see the top 15 keywords that are representative of the topic.

The `show_topics()` defined below creates that.

```
# Show top n keywords for each topic
def show_topics(vectorizer=vectorizer, lda_model=lda_model, n_words=20):
    keywords = np.array(vectorizer.get_feature_names())
    topic_keywords = []
    for topic_weights in lda_model.components_:
        top_keyword_locs = (-topic_weights).argsort()[:n_words]
        topic_keywords.append(keywords.take(top_keyword_locs))
    return topic_keywords

topic_keywords = show_topics(vectorizer=vectorizer, lda_model=best_lda_model, n_words=15)

# Topic - Keywords Dataframe
df_topic_keywords = pd.DataFrame(topic_keywords)
df_topic_keywords.columns = ['Word '+str(i) for i in range(df_topic_keywords.shape[1])]
df_topic_keywords.index = ['Topic '+str(i) for i in range(df_topic_keywords.shape[0])]
df_topic_keywords
```

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10	Word 11	Word 12	Word 13	Word 14
Topic 0	game	team	win	play	year	line	organization	subject	league	season	fan	new	san	baseball	red
Topic 1	file	window	use	program	image	run	version	line	available	server	ftp	set	user	software	display
Topic 2	say	god	people	write	think	know	believe	christian	make	subject	line	good	just	organization	thing
Topic 3	people	gun	say	article	write	just	know	time	make	think	organization	line	subject	child	year
Topic 4	key	use	space	government	make	law	line	organization	write	subject	public	people	encryption	year	know
Topic 5	line	subject	organization	post	university	host	nntp	edu	thank	write	computer	article	know	use	distribution
Topic 6	max	use	bit	line	subject	wire	chip	bhj	organization	giz	power	signal	high	circuit	cable
Topic 7	drive	good	write	line	think	organization	subject	article	scsi	year	time	make	game	play	just
Topic 8	armenian	right	people	state	government	turkish	war	write	israeli	say	israel	article	arab	muslim	subject
Topic 9	car	write	line	article	subject	organization	good	just	post	bike	nntp	host	look	think	dod

(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/t/o/s/top_15_topic_keywords.png).

Top 15 topic keywords

19. How to predict the topics for a new piece of text?

Assuming that you have already built the topic model, you need to take the text through the same routine of transformations and before predicting the topic.

For our case, the order of transformations is:

```
sent_to_words() -> lemmatization() -> vectorizer.transform() ->
best_lda_model.transform()
```

You need to apply these transformations in the same order. So to simplify it, let's combine these steps into a `predict_topic()` function.

```
# Define function to predict topic for a given text document.
nlp = spacy.load('en', disable=['parser', 'ner'])

def predict_topic(text, nlp=nlp):
    global sent_to_words
    global lemmatization

    # Step 1: Clean with simple_preprocess
    mytext_2 = list(sent_to_words(text))

    # Step 2: Lemmatize
    mytext_3 = lemmatization(mytext_2, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

    # Step 3: Vectorize transform
    mytext_4 = vectorizer.transform(mytext_3)

    # Step 4: LDA Transform
    topic_probability_scores = best_lda_model.transform(mytext_4)
    topic = df_topic_keywords.iloc[np.argmax(topic_probability_scores), :].values.tolist()
    return topic, topic_probability_scores

# Predict the topic
mytext = ["Some text about christianity and bible"]
topic, prob_scores = predict_topic(text = mytext)
print(topic)
```

```
['say', 'god', 'people', 'write', 'think', 'know', 'believe', 'christian', 'make', 'subject',
```

`mytext` has been allocated to the topic that has religion and Christianity related keywords, which is quite meaningful and makes sense.

20. How to cluster documents that share similar topics and plot?

You can use k-means clustering on the document-topic probability matrix, which is nothing but `lda_output` object. Since our best model has 15 clusters, I've set `n_clusters=15` in `KMeans()`.

Alternately, you could avoid k-means and instead, assign the cluster as the topic column number with the highest probability score.

We now have the cluster number. But we also need the X and Y columns to draw the plot.

For the X and Y, you can use SVD on the `lda_output` object with `n_components` as 2. SVD ensures that these two columns captures the maximum possible amount of information from `lda_output` in the first 2 components.

```
# Construct the k-means clusters
from sklearn.cluster import KMeans
clusters = KMeans(n_clusters=15, random_state=100).fit_predict(lda_output)

# Build the Singular Value Decomposition(SVD) model
svd_model = TruncatedSVD(n_components=2) # 2 components
lda_output_svd = svd_model.fit_transform(lda_output)

# X and Y axes of the plot using SVD decomposition
x = lda_output_svd[:, 0]
y = lda_output_svd[:, 1]

# Weights for the 15 columns of Lda_output, for each component
print("Component's weights: \n", np.round(svd_model.components_, 2))

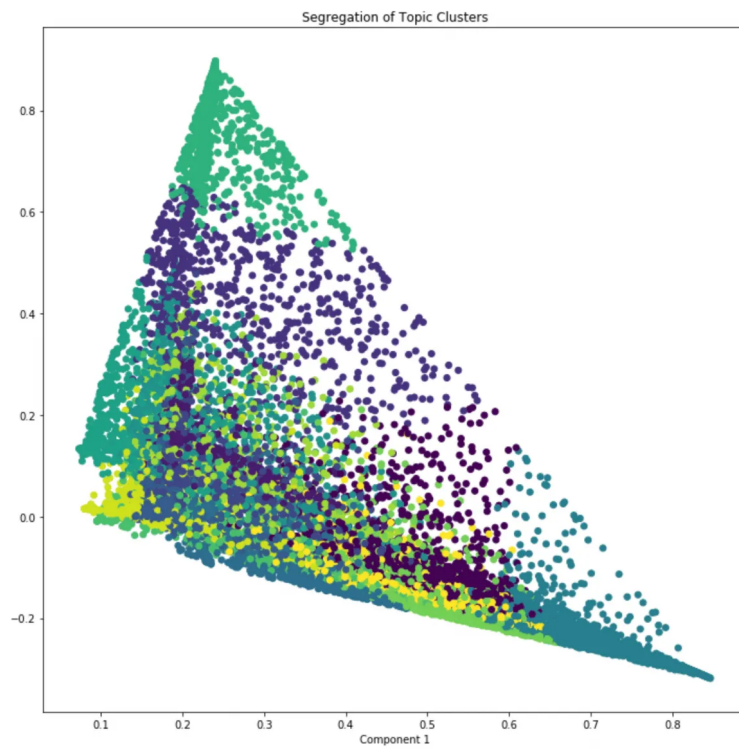
# Percentage of total information in 'Lda_output' explained by the two components
print("Perc of Variance Explained: \n", np.round(svd_model.explained_variance_ratio_, 2))
```

```
Component's weights:
[[ 0.08  0.23  0.24  0.14  0.2   0.85  0.09  0.19  0.07  0.2 ]
 [ 0.02 -0.1   0.9   0.16  0.16 -0.32 -0.01 -0.01  0.13  0.09]]
Perc of Variance Explained:
[ 0.09  0.21]
```

We have the X, Y and the cluster number for each document.

Let's plot the document along the two SVD decomposed components. The color of points represents the cluster number (in this case) or topic number.

```
# Plot
plt.figure(figsize=(12, 12))
plt.scatter(x, y, c=clusters)
plt.xlabel('Component 2')
plt.xlabel('Component 1')
plt.title("Segregation of Topic Clusters", )
```



(https://machinelearningplus.sirv.com/WP_www.machinelearningplus.com/2018/04/t/o/s/topic_clusters.png).

Topic Clusters

21. How to get similar documents for any given piece of text?

Once you know the probability of topics for a given document (using `predict_topic()`), compute the euclidean distance (<http://mathworld.wolfram.com/Distance.html>), with the probability scores of all other documents.

The most similar documents are the ones with the smallest distance.

```
from sklearn.metrics.pairwise import euclidean_distances

nlp = spacy.load('en', disable=['parser', 'ner'])

def similar_documents(text, doc_topic_probs, documents = data, nlp=nlp, top_n=5, verbose=False):
    topic, x = predict_topic(text)
    dists = euclidean_distances(x.reshape(1, -1), doc_topic_probs)[0]
    doc_ids = np.argsort(dists)[:top_n]
    if verbose:
        print("Topic KeyWords: ", topic)
        print("Topic Prob Scores of text: ", np.round(x, 1))
        print("Most Similar Doc's Probs: ", np.round(doc_topic_probs[doc_ids], 1))
    return doc_ids, np.take(documents, doc_ids)
```



```
# Get similar documents
mytext = ["Some text about christianity and bible"]
doc_ids, docs = similar_documents(text=mytext, doc_topic_probs=lda_output, documents = data, to
print('\n', docs[0][:500])
```

```
Topic KeyWords: ['say', 'god', 'people', 'write', 'think', 'know', 'believe', 'christian', 'm
Topic Prob Scores of text: [[ 0.  0.  0.8 0.  0.  0.  0.  0.  0.  0. ]]
Most Similar Doc's Probs: [[ 0.  0.  0.8 0.  0.  0.  0.1 0.  0.  0. ]]

From: Subject: about Eliz C Prophet Lines: 21 Rob Butera asks about a book called THE LOST YEA
```

22. Conclusion

We've covered some cutting-edge topic modeling approaches in this post. If you managed to work this through, well done. For those concerned about the time, memory consumption and variety of topics when building topic models check out the [gensim tutorial on LDA](https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/) (<https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/>).

I will meet you with a new tutorial next week.

Comments Community Privacy Policy Login

Recommend 1 Sort by Newest

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Monica Puerto • 2 months ago

If we are using Gensim Package for LDA and not sklearn; is the passes parameter in the LDA similar to cross validation? I landed on this article when looking for Latent Dirichlect Allocation and cross validation

^ | v • Reply •



Cloves Adriano • 4 months ago

For "13. Compare LDA Model Performance Scores" use:

```
import seaborn as sns

results = pd.DataFrame(model.cv_results_)

current_palette = sns.color_palette("Set2", 3)

plt.figure(figsize=(12,8))

sns.lineplot(data=results,
x='param_n_components',
y='mean_test_score',
hue='param_learning_decay',
palette=current_palette,
marker='o'
)

plt.show()
```

Feedback

^ | v • Reply •



Melania Castillo B • 5 months ago

Hi, thanks for this amazing work! It's helped me a lot defining details, even though I am working with R. Someone with an idea if there is an amazing tutorial like this one but using R?

^ | v • Reply •



Erick Irigoyen • 10 months ago

Great example. I was able to follow along step # 13. I keep getting the following error: `AttributeError: 'str' object has no attribute 'parameters'`. Still learning python, so any piece of advice on how to debug that error would be greatly appreciated.

^ | v • Reply •



Cloves Adriano → Erick Irigoyen • 4 months ago

For "13. Compare LDA Model Performance Scores" use:

```
import seaborn as sns

results = pd.DataFrame(model.cv_results_)

current_palette = sns.color_palette("Set2", 3)

plt.figure(figsize=(12,8))

sns.lineplot(data=results,
             x='param_n_components',
             y='mean_test_score',
             hue='param_learning_decay',
             palette=current_palette,
             marker='o'
            )

plt.show()
```

^ | v • Reply •



Zirui Hao • a year ago

Nice work!!

But, when I try to predict the new text, I get an error. `IndexError: single positional indexer is out-of-bounds`. Can someone explain this please?

3 ^ | v • Reply •



Herry Wang • a year ago

`AttributeError: 'GridSearchCV' object has no attribute 'grid_scores_'`

^ | v • Reply •



Diana Shao → Herry Wang • a year ago

`'grid_scores_'` is now `"cv_results_"`.

^ | v • Reply •



Lei • 2 years ago

The `GridSearchCV` part is not right. I tested on my dataset and it always gives me the smallest number of topics. I guess it's because likelihood should not be used as the score measure

1 ^ | v • Reply •



Alessandro Marin → Lei • 8 months ago • edited

See the answer on this [stakoverflow](#): thread and its link

There is a bug in scikit-learn causing the perplexity to increase. Apparently that is the same for likelihood scores

^ | v • Reply •



Dinesh Reddy • 2 years ago

Thanks for excellent article.

I have one doubt how to get back the main word after Lemmatization done.

for example word like systems will be system. In the final step at `df_topic_keywords` i need to get my main word instead of getting root word.

is there a way to get the main word used in the data frame

^ | v • Reply •



Mike • 2 years ago

Thank you very much for such a good tutorial. Have you done any comparison among different libraries in terms of the LDA algorithm and what one do you prefer?

^ | v • Reply •



Mike • 2 years ago

Awesome tutorial again, just like the one on Gensim LDA. My question is that "have you done any comparison between different libraries and what library do you recommend to use?

^ | v • Reply •



Selva → Mike • 2 years ago

That would be gensim.

^ | v • Reply •



Megan • 2 years ago

If I wanted to add bigrams/trigrams in here as in the Gensim model, where would I inset them? I tried to work it in just before constructing the document-word matrix, and got a bunch of errors!

^ | v • Reply •



Chauhan • 2 years ago

Thanks for the very useful article about topic modeling. It helps to understand the aspect in detail.

I have few questions. Can I use the same for Gujarati language dataset? How to make the model work with the dataset which is not in English?

^ | v • Reply •



Jeff • 2 years ago

Excellent article. I have been learning a lot. Keep up the good work. Kindly cover other Text related tasks. Thank you.

^ | v • Reply •



seongmin • 2 years ago

Thanks for this good example!

May I ask you why you created the k mean with `n_clusters = 15`? I thought the number of the topics in the model is 10.

^ | v • Reply •



Timothy Rajan • 2 years ago

Excellent article and learnt a lot from this article. Thanks a lot for a detailed article.

Quick question. I created a project by following your code. I am able to get a good topic representation for a given data set. This document is about 2000 words and lets call it doc A. Now I have 10 document with average words around 2000 in each of the document (Similar to doc A) . I believe I need to follow the steps mentioned in Section 19 .(How to predict the topics for a new piece of text?) to get the topics for the 10 documents.

Here my question is

Are we reusing the same model (best_lda_model) for the remaining 10 docs.

If yes, how did "Some text about christianity and bible" as input to best_lda_model produced the topic.

[see more](#)

^ | v • Reply •



Chris Palmer • 2 years ago

Hi Ivyswius

I observed this too, its not really appropriate to measure likelihood, as you've seen (and I did too) its increasingly likely you will find a place in a topic as you reduce the number. I think Selva needs to re-consider his advice here. With the Gensim tutorial he uses Coherence which delivers a proper measure - but I've spent some time looking for a way to do this with sklearn and haven't found one yet. In the end the topic modelling capabilities of Gensim or Mallet in the other tutorial seem more fit for purpose.

^ | v • Reply •



Ivyswius • 2 years ago

Hi Selva,

Thanks for this excellent tutorial! It is really helpful! While I was using GridSearchCV(), the optimal number of topics will finally converge to 1, which does not really make sense. And I see that in your output, 10 is optimal within [10, 15, 20, 25, 30]. So what will happen if continuing exploring (e.g. [1, 3, 5, 7, 10])? Will it converge to 1?

Thanks!

^ | v • Reply •



Alessandro Marin → Ivyswius
• 8 months ago • edited

likelihood scores and perplexity seem to be buggy in sklearn! See my comment above

^ | v • Reply •



Selva Prabhakaran → Ivyswius • 2 years ago

This is a subjective matter. Some pointers: How many topics do you think are there in your document? Are they different from each other or are they very similar?

Secondly, you don't have to set number of topics as 1. In the approach discussed, in sklearn related topics tend to get clubbed together as one. If you want to keep them as separate, check out [gensim](#).

With multiple approaches, more the control you have.

^ | v • Reply •



Chris Palmer • 2 years ago

I ran out of memory creating the dense matrix for measuring sparsity. So experimented and found it is not necessary to create a dense matrix to measure sparsity - just use the properties of the sparse matrix:

shape : 2-tuple

Shape of the matrix

nnz

Number of nonzero elements

```
print("Sparsity: ", (data_vectorized.nnz /  
(data_vectorized.shape[0] *
```

```
data_vectorized.shape[1]))*100, "%")
```

^ | v • Reply •



Gleb • 2 years ago

Thanks for sharing this. It is a great article. I am relatively new to python. I followed your code but i got stuck getting the 30 most common terms in the whole text with their weights for each topic.

^ | v • Reply •



pronoy • 2 years ago

Thanks a lot selva learnt a lot of libraries and their uses i have one question post the prediction of topic number how are u telling which all doc from the original file belong to a particular topic as we have only the doc number in our output and we are assigning random state

^ | v • Reply •



Selva Prabhakaran → **pronoy** • 2 years ago

The topic that contributes to the most proportion to the document can be taken as the dominant topic for that doc. Based on your requirement, you can take more than 1 topic, perhaps the top 3 maybe.

^ | v • Reply •



Xiaowei MA • 2 years ago

Dear Selva,

Thanks for such a great tutorial. I ran the codes within the article, but ran into some problems at some steps. For instance, I printed out the value of `best_lda_model.n_topics`, which is 'None', (empty value). I am curious how this parameter is set, is it set during the parameter tuning step?

Besides, I also was not able to load the model 'en' from spacy, but I searched on Internet and got a workable fix. I am providing my fix here, because it may be useful to others. First, to visit the following page:
<https://github.com/explosio...>
and then download `en_core_web_sm-2.0.0.tar.gz` on the page. Next, to install the package using:
`pip install en_core_web_sm-2.0.0.tar.gz`.

^ | v • Reply •



Sista → **Xiaowei MA** • 7 months ago

I have installed the 'en' model from this using the method you mentioned here. I was successfully able to install it. But, my python environment is not recognizing it, it is still giving error report saying that it can't find the 'en' model. Also, I am using python 2.7 and running the code on MAC OS, do you have suggestions for resolving this situation

^ | v • Reply •



Selva Prabhakaran → **Xiaowei MA** • 2 years ago

Thanks Xiaowei.

Its probably got to do with how you set up your search parameters when you define `search_params`.

```
search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}
```

In this case, LDA will grid search for `n_components` (or `n_topics`) as 10, 15, 20, 25, 30.

Also, check if your corpus is intact inside

data_vectorized just before starting
model.fit(data_vectorized). To be sure, run
`data_dense = data_vectorized.todense()` and
check few rows of `data_dense`.

^ | v • Reply •



Tano → Selva Prabhakaran • 2 years ago

Hi Selva,
great tutorial indeed!
I have got the same problem, but it
depends on sklearn:
<http://scikit-learn.org/sta...>

`n_topics : int, optional (default=None)`

`This parameter has been renamed to
n_components and will be removed in
version 0.21. .. deprecated:: 0.19`

n_topics is still working, but deprecated
when running the model, but it is not
working anymore when it is called later.
just swap `n_topics` with
`n_components` and it works

^ | v • Reply •



Selva Prabhakaran → Tano

• 2 years ago

Thank you for the tip

Copyright Machine Learning Plus (<https://www.machinelearningplus.com/>). All rights reserved.

[Home \(https://www.machinelearningplus.com\)](https://www.machinelearningplus.com) [Contact Us \(https://www.machinelearningplus.com/contact-us/\)](https://www.machinelearningplus.com/contact-us/)

[Privacy Policy \(https://www.machinelearningplus.com/privacy-policy/\)](https://www.machinelearningplus.com/privacy-policy/) [About Selva \(https://www.machinelearningplus.com/about/\)](https://www.machinelearningplus.com/about/)

[Web Site Terms and Conditions of Use \(https://www.machinelearningplus.com/terms-of-use/\)](https://www.machinelearningplus.com/terms-of-use/)