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

*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.*

**Contents**

[1. Introduction](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#1introduction)  
[2. Load the packages](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#2loadthepackages)  
[3. Import Newsgroups Text Data](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#3importnewsgroupstextdata)  
[4. Remove emails and newline characters](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#4removeemailsandnewlinecharacters)  
[5. Tokenize and Clean-up using gensim’s simple\_preprocess()](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#5tokenizeandcleanupusinggensimssimple_preprocess)  
[6. Lemmatization](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#6lemmatization)  
[7. Create the Document-Word matrix](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#7createthedocumentwordmatrix)  
[8. Check the Sparsicity](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#8checkthesparsicity)  
[9. Build LDA model with sklearn](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#9buildldamodelwithsklearn)  
[10. Diagnose model performance with perplexity and log-likelihood](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#10diagnosemodelperformancewithperplexityandloglikelihood)  
[11. How to GridSearch the best LDA model?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#11howtogridsearchthebestldamodel)  
[12. How to see the best topic model and its parameters?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#12howtoseethebesttopicmodelanditsparameters)  
[13. Compare LDA Model Performance Scores](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#13compareldamodelperformancescores)  
[14. How to see the dominant topic in each document?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#14howtoseethedominanttopicineachdocument)  
[15. Review topics distribution across documents](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#15reviewtopicsdistributionacrossdocuments)  
[16. How to visualize the LDA model with pyLDAvis?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#16howtovisualizetheldamodelwithpyldavis)  
[17. How to see the Topic’s keywords?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#17howtoseethetopicskeywords)  
[18. Get the top 15 keywords each topic](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#18getthetop15keywordseachtopic)  
[19. How to predict the topics for a new piece of text?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#19howtopredictthetopicsforanewpieceoftext)  
[20. How to cluster documents that share similar topics and plot?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#20howtoclusterdocumentsthatsharesimilartopicsandplot)  
[21. How to get similar documents for any given piece of text?](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#21howtogetsimilardocumentsforagivendocument)  
[22. Conclusion](https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/#21conclusion)



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/). In this tutorial, however, I am going to use python’s the most popular machine learning library – [scikit learn](http://scikit-learn.org/stable/" \t "_blank).

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 matplotlibfor visualization and numpy and pandas for manipulating and viewing data in tabular format.

Let’s import them.

# 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).

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'

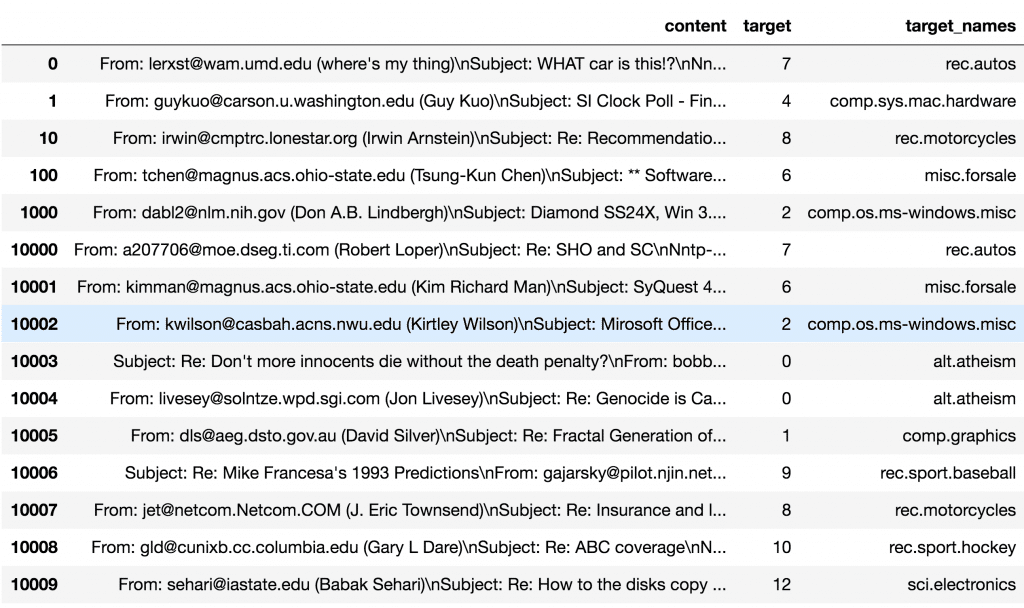
'rec.sport.baseball' 'rec.sport.hockey' 'sci.electronics' 'sci.space'

'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)

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/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/).

# 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.

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 punctuations

data\_words = list(sent\_to\_words(data))

print(data\_words[:1])

[['from', 'wheres', 'my', 'thing', 'subject', 'what', 'car', 'is', 'this', 'nntp', 'posting', 'host', 'rac', 'wam', 'umd', 'edu', 'organization', 'university', 'of', 'maryland', 'college', 'park', 'lines', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'enlighten', 'me', 'on', 'this', 'car', 'saw', 'the', 'other', 'day', (..truncated..)]]

**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"""

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 if token.pos\_ in allowed\_postags]))

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 maryland college park line be wonder anyone out there could enlighten car see other days be door sport car look be late early be call bricklin door be really small addition front bumper be separate rest body be know anyone can tellme model name engine spec year production where car be make history whatev info have funky look car mail thank bring neighborhood lerxst' (..truncated..)]

**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 occurences of a word

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 iter

evaluate\_every = -1, # compute perplexity every n iters, default: Don't

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. \* log-likelihood per word)) is considered to be good. Let’s check for our model.

# Log Likelyhood: 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 Likelyhood 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\_likelyhoods\_5 = [round(gscore.mean\_validation\_score) for gscore in model.grid\_scores\_ if gscore.parameters['learning\_decay']==0.5]

log\_likelyhoods\_7 = [round(gscore.mean\_validation\_score) for gscore in model.grid\_scores\_ if gscore.parameters['learning\_decay']==0.7]

log\_likelyhoods\_9 = [round(gscore.mean\_validation\_score) for gscore in model.grid\_scores\_ if gscore.parameters['learning\_decay']==0.9]

# Show graph

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

plt.plot(n\_topics, log\_likelyhoods\_5, label='0.5')

plt.plot(n\_topics, log\_likelyhoods\_7, label='0.7')

plt.plot(n\_topics, log\_likelyhoods\_9, label='0.9')

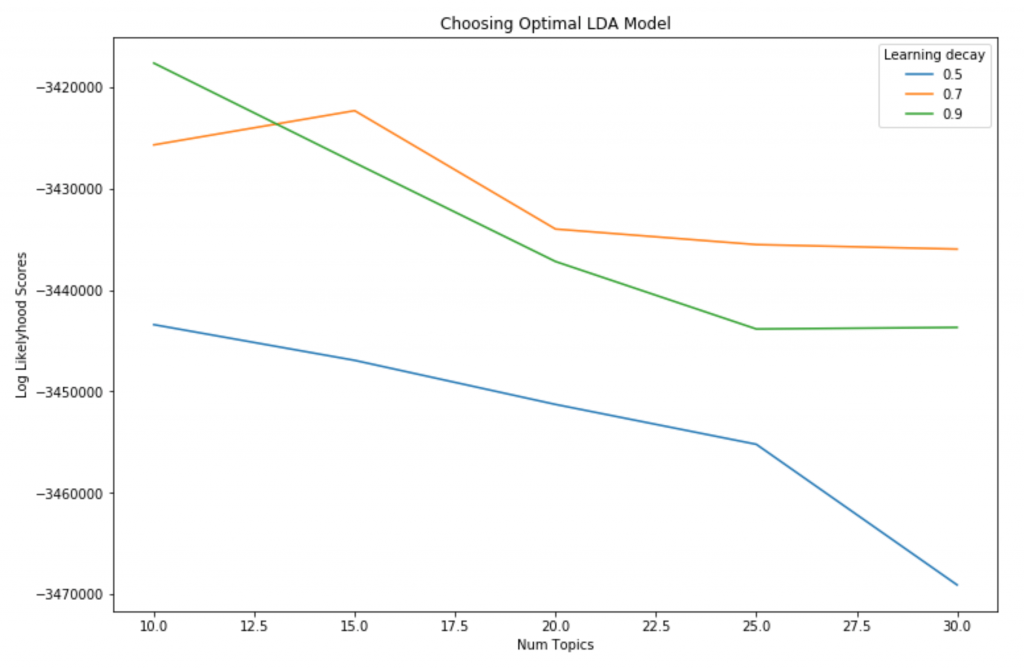
plt.title("Choosing Optimal LDA Model")

plt.xlabel("Num Topics")

plt.ylabel("Log Likelyhood Scores")

plt.legend(title='Learning decay', loc='best')

plt.show()

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/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):

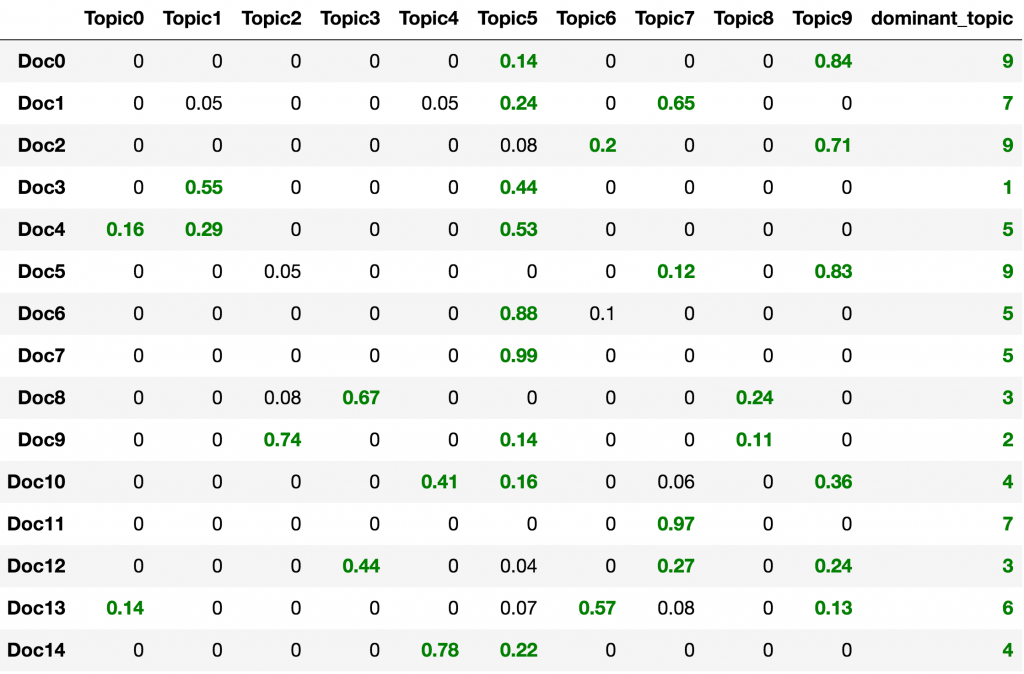
weight = 700 if val > .1 else 400

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

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/document_topic_weights.png)

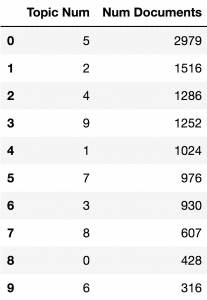
DOCUMENT TOPIC WEIGHTS: DF\_DOCUMENT\_TOPICS

**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

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/topic_document_distribution.png)

TOPIC DOCUMENT DISTRIBUTION: DF\_TOPIC\_DISTRIBUTION

**16. How to visualize the LDA model with pyLDAvis?**

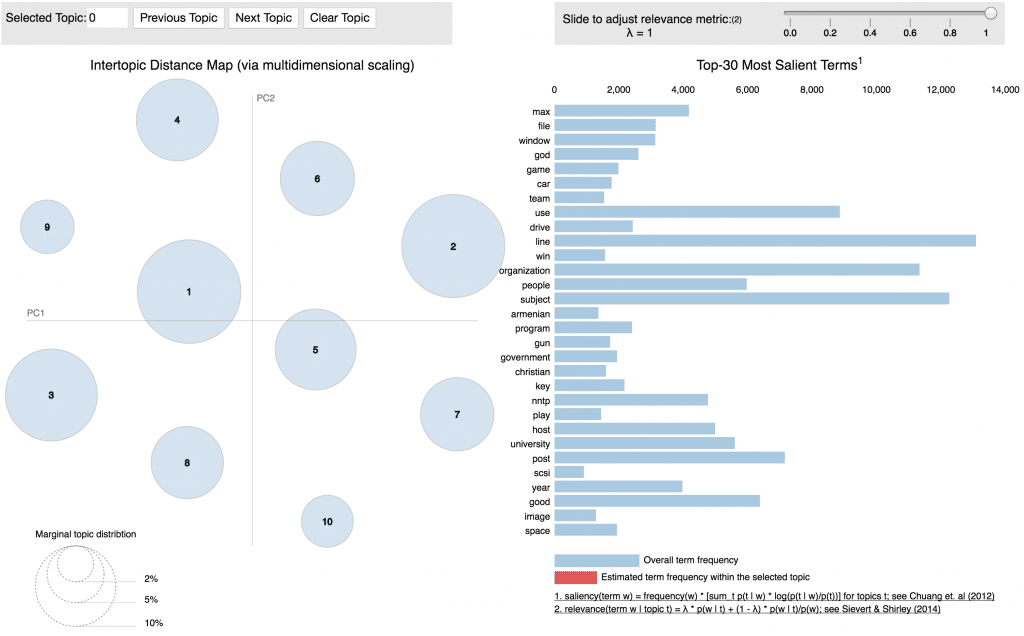
The pyLDAvis offers the best visualization to view the topics-keywords distribution.

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://www.machinelearningplus.com/wp-content/uploads/2018/04/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\_)

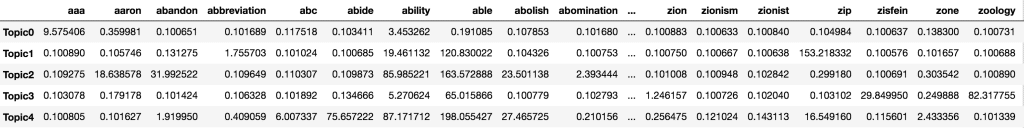
# Assign Column and Index

df\_topic\_keywords.columns = vectorizer.get\_feature\_names()

df\_topic\_keywords.index = topicnames

# View

df\_topic\_keywords.head()

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/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)

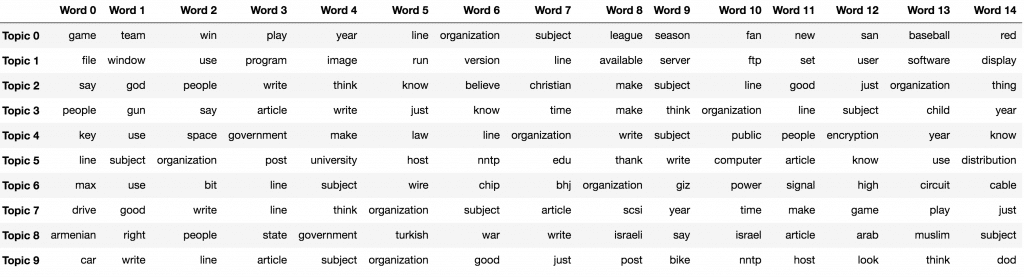
# Topic - Keywords Dataframe

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

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/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', 'line', 'good', 'just', 'organization', 'thing']

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 probabilioty matrix, which is nothing but lda\_output object. Since out 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

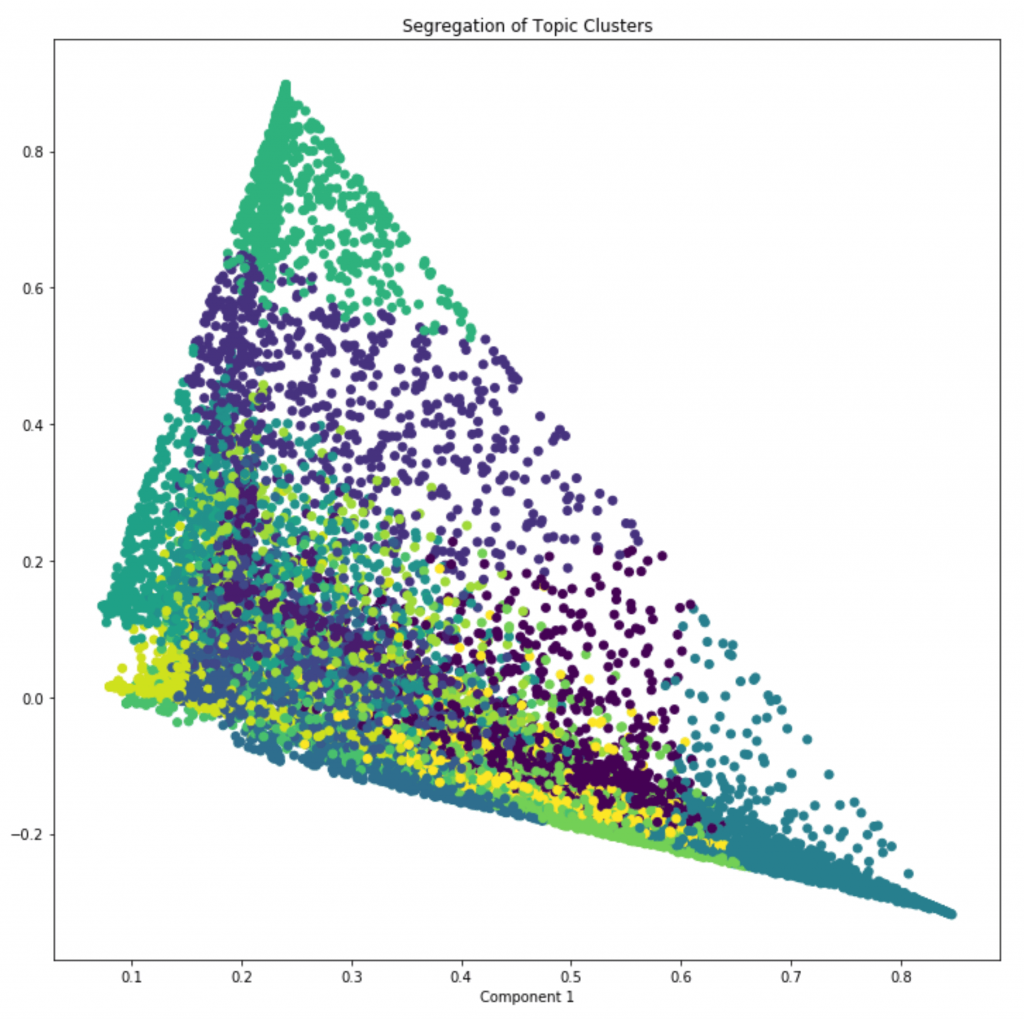
plt.figure(figsize=(12, 12))

plt.scatter(x, y, c=clusters)

plt.xlabel('Component 2')

plt.xlabel('Component 1')

plt.title("Segregation of Topic Clusters", )

[](https://www.machinelearningplus.com/wp-content/uploads/2018/04/topic_clusters.png)

TOPIC CLUSTERS

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

Once you know the probaility 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, top\_n=1, verbose=True)

print('\n', docs[0][:500])

Topic KeyWords: ['say', 'god', 'people', 'write', 'think', 'know', 'believe', 'christian', 'make', 'subject', 'line', 'good', 'just', 'organization', 'thing']

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 YEARS OF JESUS, by Elizabeth Clare Prophet. I do not know the book. However, Miss Prophet is the leader of a group (The Church Universal and Triumphant) derived from the I AM group founded by a Mr. Ballard who began his mission in the 1930s (I am writing this from memory and may not have all the details straight -- for an old account, check your library for a book by Marcus Bach) after an eighteenth-centu

**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/).

I will meet you with a new tutorial next week.