```
import sys
import os


current_work_directory = os.getcwd()    # Return a string representing the current working
print('Current work directory: {}'.format(current_work_directory))
# Make sure it's an absolute path.
abs_work_directory = os.path.abspath(current_work_directory)
print('Current work directory (full path): {}'.format(abs_work_directory))
print()

filename = 'subset.csv'
# Check whether file exists.
if not os.path.isfile(filename):
    # Stop with leaving a note to the user.
    print('It seems file "{}" not exists in directory: "{}"'.format(filename, current_work_
    sys.exit(1)

import csv

with open('subset.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        print(row)
```

```
Current work directory: C:\Users\shabe
Current work directory (full path): C:\Users\shabe

['address', 'attributes', 'business_id', 'categories', 'city', 'hours', 'is_open', '
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['631 Bloor St W', '{\'BusinessParking\': "{\'garage\': False, \'street\': False, \'
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['3417 Derry Road E, Unit 103', '{\'Alcohol\': \'none\', \'BusinessAcceptsCreditCard
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
['4568 Highway 7 E', "{'GoodForKids': 'True', 'NoiseLevel': 'loud', 'RestaurantsAtti
```

```python
import pandas as pd
from pandas import DataFrame

ReadCsv = pd.read_csv (r'C:\Users\shabe\subset.csv')

df = DataFrame(ReadCsv,columns=['address', 'attributes', 'business_id', 'categories', 'city

print (df)


from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

no_features = 1000

# NMF is able to use tf-idf
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=no_features, stop_wo
tfidf = tfidf_vectorizer.fit_transform(df['text'])
tfidf_feature_names = tfidf_vectorizer.get_feature_names()

# LDA can only use raw term counts for LDA because it is a probabilistic graphical model
tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2, max_features=no_features, stop_words
tf = tf_vectorizer.fit_transform(df['text'])
tf_feature_names = tf_vectorizer.get_feature_names()
```

```
                              address  \
0                     631 Bloor St W
1                     631 Bloor St W
2                     631 Bloor St W
3                     631 Bloor St W
4                     631 Bloor St W
5                     631 Bloor St W
6                     631 Bloor St W
7         3417 Derry Road E, Unit 103
8         3417 Derry Road E, Unit 103
9         3417 Derry Road E, Unit 103
10        3417 Derry Road E, Unit 103
11        3417 Derry Road E, Unit 103
12        3417 Derry Road E, Unit 103
13        3417 Derry Road E, Unit 103
14                  4568 Highway 7 E
15                  4568 Highway 7 E
16                  4568 Highway 7 E
17                  4568 Highway 7 E
18                  4568 Highway 7 E
19                  4568 Highway 7 E
20                  4568 Highway 7 E
21                  4568 Highway 7 E
22                  4568 Highway 7 E
23                  4568 Highway 7 E
24                  4568 Highway 7 E
25                  4568 Highway 7 E
26                595 Markham Street
27                595 Markham Street
28                595 Markham Street
29                595 Markham Street
...                               ...
706701              3199 Dufferin St
706702              3199 Dufferin St
706703              3199 Dufferin St
706704              3199 Dufferin St
706705              3199 Dufferin St
706706              3199 Dufferin St
706707              3199 Dufferin St
706708              3199 Dufferin St
706709              3199 Dufferin St
706710              3199 Dufferin St
706711           2215 Queen Street E
706712           2215 Queen Street E
706713           2215 Queen Street E
706714  1402 Queen Street E, Suite D
```

```
from sklearn.decomposition import NMF, LatentDirichletAllocation

no_topics = 20

# Run NMF
nmf = NMF(n_components=no_topics, random_state=1, alpha=.1, l1_ratio=.5, init='nndsvd').fit

# Run LDA
lda = LatentDirichletAllocation(n_topics=no_topics, max_iter=5, learning_method='online', l
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\sklearn\decomposition\online_lda.py:294:
  DeprecationWarning)
```

```
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print ("Topic %d:" % (topic_idx))
        print (" ".join([feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))

no_top_words = 10
display_topics(nmf, tfidf_feature_names, no_top_words)
display_topics(lda, tf_feature_names, no_top_words)
```

```
Topic 0:
time order minutes asked got said told service wait didn
Topic 1:
chicken fried rice sauce jerk curry butter spicy wings ordered
Topic 2:
great service atmosphere selection spot awesome beer amazing prices patio
Topic 3:
food service restaurant quality excellent fast price better chinese indian
Topic 4:
sushi sashimi rolls roll salmon fresh ayce fish quality japanese
Topic 5:
good pretty service price overall bit prices selection decent nice
```

INTRODUCTION

One of the primary applications of natural language processing is to automatically extract what topics people are discussing from large volumes of text. Some examples of large text could be feeds from social media, customer reviews of hotels, movies, etc, user feedbacks, news stories, e-mails of customer complaints etc. Knowing what people are talking about and understanding their problems and opinions is highly valuable to businesses, administrators, political campaigns. It is really hard to manually read through such large volumes and compile the topics, and we need an algorithm that can read through the text doucments and output the topics discussed. I used the Latent Dirichlet Allocation (LDA) from Gensim package to perform Topic Modeling. Latent Dirichlet Allocation is the most common technique used in Topic Modeling, it is a generalized form of probabilistic latent semantic analysis (PLSA). In a nutshell, LDA considers each document as a collection of topics in a certain proportion, and each topic as a collection of keywords in a certain proportion. Once the algorithm is provided with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution. A topic is nothing but a collection of dominant keywords that are typical representatives. A Topic is identified by looking at the keywords.

We will need the stopwords from NLTK and spacy's en model for text pre-processing. We will be using the spacy model for lemmatization. Lemmatization is nconverting a word to its root word.

The core packages used are re, gensim, spacy and pyLDAvis. Besides this we will also using matplotlib, numpy and pandas for data handling and visualization.

```
Topic 0:
```

```
import nltk; nltk.download('stopwords')

import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
```

```
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shabe\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\Users\shabe\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detect
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

We will be using Yelp reviews from Toronto, and see what topics are within those reviews. Pandas was used to import the data.

```
# NLTK Stop words
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

```
# Import Dataset

import pandas as pd
from pandas import DataFrame

ReadCsv = pd.read_csv (r'C:\Users\shabe\subset.csv')

df = DataFrame(ReadCsv,columns=['address', 'attributes', 'business_id', 'categories', 'city

#print (df)

#print(df.target_names.unique())
#df.head()
```

Next, we remove unwanted spaces and characters using regular expression. Below is the code to do it. The text still looks messy even after removing unwanted characters and extra spaces. It is not ready for the LDA to consume. We need to break down each sentence into a list of words through tokenization, while clearing up all the messy text in the process. We will use Gensim's simple_preprocess is great for this, and we removed puncuation as well.

```
# Convert to list
data = df.text.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])
```

```
['Hallelujah! I FINALLY FOUND IT! The frozen yogurt that launched the Red '
 'Mango and Pinkberry craze in the States. (Google it.) The Canadian '
 'incarnation goes by the name Yogoberri and I discovered it inside a tiny '
 'Korean bakery along Bloor Streets K-town. For the uninitiated, this frozen '
 'yogurt is more tart and less sweet than the TCBY kind. You can have plain '
 'vanilla yogurt but its all about the toppings; fresh fruit, nuts, '
 'cereal...weird-looking powders I never tried. A small (5 oz.) is $2.97 + 95 '
 'cents per topping. Medium (8 oz.) including three toppings is $4.99. I used '
 'to eat this frozen yogurt all the time when I lived in Korea and I was '
 'practically weeping with joy when I was reunited with it today. Shameless '
 'plea: Go eat lots of it so the chain will multiply and open a branch near my '
 'home. THANKS! (FYI, the 5 stars is for the yogurt. I havent tried anything '
 'else at the bakery.) **ETA: Dear Fro Yo Gods, Thanks for opening up '
 'Blushberry closer to my home. xoxo, susan c.**']
```

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

data_words = list(sent_to_words(data))

print(data_words[:1])
```

```
[['hallelujah', 'finally', 'found', 'it', 'the', 'frozen', 'yogurt', 'that', 'launch
```

After tokenization and removing puncuation, we created Bigrams and Trigrams. Bigrams are two words frequently occurring together in the document. Trigrams are 3 words frequently occurring.

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 to bigrams. once the bigrams model is ready, we are going to define the functions to remove the stopwords, make bigrams and lemmatization and call them. We would also need to create the inputs for LDA. The two main inputs to the LDA topic model are the dictionary(id2word) and the corpus. These are created later down in the code.

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

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\gensim\models\phrases.py:494: UserWarning
  warnings.warn("For a faster implementation, use the gensim.models.phrases.Phraser
['hallelujah', 'finally', 'found', 'it', 'the', 'frozen_yogurt', 'that', 'launched',
```

```
# 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 do

def make_bigrams(texts):
```

```
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[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


# 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)
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

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

print(data_lemmatized[:1])
```

```
[['finally', 'find', 'frozen_yogurt', 'launch', 'red', 'mango', 'pinkberry', 'craze'
```

This is where the inputs for LDA is created. Gensim creates a unique id for each word in the document. The produced corpus shown below is a mapping of (word_id, word_frequency). This is used as the input by the LDA model.

If you want to see what word a given id corresponds to, pass the id as a key to the dictionary. or look at the Human readable format.

```
# 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])
```

```
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (1
```

```
id2word[0]
```

```
'anything'
```

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('anything', 1),
  ('bakery', 2),
  ('bloor', 1),
  ('blushberry', 1),
  ('branch', 1),
  ('canadian', 1),
  ('cent', 1),
  ('cereal', 1),
  ('chain', 1),
  ('close', 1),
  ('craze', 1),
  ('dear_fro', 1),
  ('discover', 1),
  ('eat', 2),
  ('else', 1),
  ('eta', 1),
  ('finally', 1),
  ('find', 1),
  ('fresh', 1),
  ('frozen_yogurt', 3),
  ('fruit', 1),
  ('fyi', 1),
  ('go', 2),
  ('god', 1),
  ('have', 1),
  ('home', 2),
  ('incarnation', 1),
  ('include', 1),
  ('joy', 1),
  ('kind', 1),
  ('korea', 1),
  ('korean', 1),
  ('launch', 1),
  ('less', 1),
  ('live', 1),
  ('look', 1),
  ('lot', 1),
  ('mango', 1),
  ('medium', 1),
  ('name', 1),
  ('never', 1),
  ('not', 1),
  ('nut', 1),
  ('open', 2),
  ('pinkberry', 1),
  ('plain', 1),
  ('plea', 1),
  ('powder', 1),
  ('practically', 1),
  ('red', 1),
  ('reunite', 1),
  ('shameless', 1),
  ('small', 1),
  ('star', 1),
  ('state', 1),
  ('street', 1),
  ('susan', 1),
```

```
('sweet', 1),
('tant', 1)
```

We have everything required to train the LDA model. In addition to the corpus and dictionary,we need to provide the number of topics as well. alpha and eta are hyperparameters that affect sparsity of the topics. According to the Gensim docs, both defaults to 1.0/num_topics prior. Chunksize is the number of documents to be used in each training chunk,and update_every determines how often the model parameters should be updated and passes is the total number of training passes.

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=20,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

The above LDA model is built with 20 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic. We can see the keywords for each topic and the weightage(importance) of each keyword using lda_model.print_topics() as shown next. Topic 0 is a represented as 0.024*"day" + 0.023*"ask" + 0.020*"work" + 0.020*"tell" + 0.019*"need" + '0.018*"say" + 0.018*"never" + 0.017*"hour" + 0.016*"customer" + '0.016*"leave. It means the top 10 keywords that contribute to this topic are: ‘day’, ‘ask’, ‘work’.. and so on and the weight of ‘day’ on topic 0 is 0.024. The weights reflect how important a keyword is to that topic.

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

```
[(0,
  '0.024*"day" + 0.023*"ask" + 0.020*"work" + 0.020*"tell" + 0.019*"need" + '
  '0.018*"say" + 0.018*"never" + 0.017*"hour" + 0.016*"customer" + '
  '0.016*"leave"'),
 (1,
  '0.040*"chicken" + 0.032*"taste" + 0.030*"fry" + 0.027*"sauce" + '
  '0.021*"soup" + 0.020*"noodle" + 0.019*"pork" + 0.018*"hot" + 0.018*"meat" + '
  '0.018*"side"'),
 (2,
  '0.087*"raman" + 0.078*"store" + 0.065*"item" + 0.051*"sandwich" + '
  '0.050*"shop" + 0.023*"sell" + 0.022*"product" + 0.022*"bean" + '
  '0.020*"vegetarian" + 0.020*"vegan"'),
 (3,
  '0.033*"fancy" + 0.023*"event" + 0.019*"chat" + 0.017*"member" + '
  '0.017*"chill" + 0.016*"moment" + 0.016*"shopping" + 0.015*"management" + '
  '0.013*"push" + 0.013*"team"'),
 (4,
  '0.204*"s" + 0.076*"there" + 0.063*"that" + 0.017*"patient" + 0.016*"brand" '
  '+ 0.015*"office" + 0.015*"mistake" + 0.013*"park" + 0.012*"what" + '
  '0.012*"depend"'),
 (5,
  '0.080*"roll" + 0.077*"sushi" + 0.063*"fish" + 0.037*"thick" + '
  '0.031*"salmon" + 0.027*"piece" + 0.027*"thin" + 0.018*"easily" + '
  '0.016*"addition" + 0.016*"basic"'),
 (6,
  '0.079*"not" + 0.054*"get" + 0.053*"go" + 0.045*"be" + 0.044*"do" + '
  '0.040*"would" + 0.026*"make" + 0.023*"have" + 0.019*"look" + 0.017*"want"'),
 (7,
  '0.054*"room" + 0.030*"parking" + 0.024*"class" + 0.022*"drive" + '
  '0.019*"fix" + 0.016*"wall" + 0.016*"complaint" + 0.014*"schnitzel" + '
  '0.013*"tire" + 0.012*"crazy"'),
 (8,
  '0.113*"pizza" + 0.108*"cheese" + 0.061*"bread" + 0.046*"bowl" + '
  '0.035*"topping" + 0.033*"slice" + 0.032*"tomato" + 0.022*"avocado" + '
  '0.022*"waffle" + 0.019*"burrito"')]
```

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the mode

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized, dictionary=id2
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -7.610246942214927

Coherence Score:  0.4172512043387064
```

Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is. Topic Coherence is a measure used to evaluate topic models.Each such generated topic consists of words, and the topic coherence is applied to the top N words from the topic. It is defined as the average / median of the pairwise word-similarity scores of the words in the topic (e.g. PMI). A good model will generate coherent topics, and can be described by a short label, therefore this is what the topic coherence measure should capture

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
C:\Users\shabe\Anaconda3\lib\site-packages\pyLDAvis\_prepare.py:257: FutureWarning:
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Selected Topic: 0 Previous Topic Next Topic Clear Topic