Data Mining Project Version 1 Comments

November 17, 2019

[1]:
```
import time
s = time.time()
```

[2]:
```
import pandas as pd
import nltk; nltk.download('stopwords')
nltk.download('punkt')

from nltk.tokenize import word_tokenize

# Regular Expressions
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)
```

```
# from ngram import NGram
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/tristandealwis/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/tristandealwis/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[3]:
```
listings = pd.read_csv("listings.csv", usecols=['id', 'name', 'description',␣
↪'neighbourhood_cleansed', 'zipcode', 'property_type', 'room_type',
                                                'price', 'availability_365'])
listings = listings.rename(columns={'id': 'listing_id'})

reviews = pd.read_csv("reviews.csv", usecols=['listing_id', 'comments'])
```

[4]:
```
df = pd.merge(listings, reviews, on='listing_id')
```

[5]:
```
df.describe();
```

[6]:
```
df.head();
```

[7]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
```

[8]:
```
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(df['comments']))
```

[9]:
```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=100) # higher␣
↪threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=100)

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

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

```
['lisa', 'is', 'wonderful', 'kind', 'and', 'thoughtful', 'host', 'the',
```

```
'listing', 'is', 'accurate', 'there', 'is', 'an', 'entire', 'floor', 'for',
'the', 'guest', 'with', 'full', 'bath', 'and', 'kitchen', 'area', 'looking',
'out', 'over', 'leafy', 'backyard', 'everything', 'needed', 'for', 'my', 'one',
'month', 'stay']
```

[10]:
```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in␣
 ↪stop_words] for doc in texts]

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

def 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
```

[11]:
```
# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)
# print('type(data_words_nostops): ', type(data_words_nostops))

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)
# print('type(data_words_bigrams): ', type(data_words_bigrams))
# print(data_words_bigrams)

# Form Trigrams
data_words_trigrams = make_trigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# ! python3 -m spacy download en

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

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

print(data_lemmatized[:1])
```

```
[['wonderful', 'kind', 'thoughtful', 'host', 'list', 'accurate', 'entire',
'floor', 'guest', 'full', 'area', 'look', 'backyard', 'need', 'month', 'stay']]
```

[12]:
```
# 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, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9,
1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1), (15, 1)]]
```

[13]:
```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=5,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha=2,
                                            per_word_topics=True)
```

[14]:
```
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.037*"need" + 0.034*"time" + 0.031*"make" + 0.030*"place" + '
  '0.026*"amazing" + 0.023*"go" + 0.021*"get" + 0.019*"come" + '
  '0.018*"experience" + 0.018*"back"'),
 (1,
  '0.045*"room" + 0.023*"bed" + 0.018*"day" + 0.017*"night" + 0.015*"bathroom" '
  '+ 0.011*"little" + 0.011*"people" + 0.011*"bedroom" + 0.011*"arrival" + '
  '0.010*"small"'),
 (2,
  '0.070*"nice" + 0.069*"good" + 0.048*"really" + 0.042*"close" + 0.040*"walk" '
  '+ 0.032*"subway" + 0.029*"area" + 0.028*"neighborhood" + 0.026*"restaurant" '
  '+ 0.023*"lot"'),
 (3,
  '0.158*"great" + 0.138*"place" + 0.101*"host" + 0.078*"location" + '
  '0.058*"recommend" + 0.039*"easy" + 0.039*"clean" + 0.032*"super" + '
```

```
 '0.025*"check" + 0.023*"highly"'),
(4,
 '0.152*"stay" + 0.061*"apartment" + 0.060*"would" + 0.054*"clean" + '
 '0.043*"comfortable" + 0.040*"definitely" + 0.031*"perfect" + 0.028*"space" '
 '+ 0.026*"well" + 0.025*"feel"')]
```

[15]:
```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how␣
 ↪good the model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized,␣
 ↪dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -6.685002498591063

Coherence Score:  0.6029178159962516
```

[18]:
```
print((s-time.time())/60/60)
```

```
-3.6143536453114615
```

[19]:
```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
/Users/tristandealwis/opt/anaconda3/lib/python3.7/site-
packages/pyLDAvis/_prepare.py:257: FutureWarning: Sorting because non-
concatenation axis is not aligned. A future version
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))
```

[19]:
```
PreparedData(topic_coordinates=              x         y  topics  cluster
Freq
topic
1      0.144291  0.075464       1        1  22.800951
0      0.046744 -0.216189       2        1  22.149307
4     -0.189675  0.357176       3        1  19.811062
2      0.298516  0.003435       4        1  18.934231
```

```
3     -0.299875 -0.219886       5        1  16.304443, topic_info=    Category
Freq        Term          Total  loglift  logprob
13   Default  672416.000000        stay  672416.000000  30.0000  30.0000
60   Default  576587.000000       great  576587.000000  29.0000  29.0000
34   Default  649985.000000       place  649985.000000  28.0000  28.0000
7    Default  369278.000000        host  369278.000000  27.0000  27.0000
30   Default  283254.000000    location  283254.000000  26.0000  26.0000
..       …            …         …           …      …      …
405   Topic5   54375.394531  convenient   66439.000000   1.6134  -4.2035
224   Topic5   91214.937500       check  159133.437500   1.2572  -3.6861
22    Topic5  140231.062500       clean  395810.281250   0.7761  -3.2561
226   Topic5   22740.662109     exactly   31957.644531   1.4735  -5.0752
439   Topic5   26630.412109    question   51013.289062   1.1637  -4.9173

[225 rows x 6 columns], token_table=      Topic      Freq         Term
term
479       3  0.999957   absolutely
16        1  0.095785       access
16        4  0.000467       access
16        5  0.903723       access
17        2  0.009325  accommodate
…     …       …           …
218       3  0.935612         well
15        3  0.999980    wonderful
368       1  0.999974         work
73        1  0.010003        would
73        3  0.989994        would

[229 rows x 3 columns], R=30, lambda_step=0.01, plot_opts={'xlab': 'PC1',
'ylab': 'PC2'}, topic_order=[2, 1, 5, 3, 4])
```

[ ]: