05_lda_with_gensim

September 29, 2021

1 Topic Modeling: Latent Dirichlet Allocation with gensim

Gensim is a specialized NLP library with a fast LDA implementation and many additional features. We will also use it in the next chapter on word vectors (see the notebook lda_with_gensim for details.

1.1 Imports & Settings

[1]:
```
import warnings
from collections import OrderedDict
from pathlib import Path

import numpy as np
import pandas as pd

# Visualization
from ipywidgets import interact, FloatSlider
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns

import pyLDAvis
from pyLDAvis.sklearn import prepare

from wordcloud import WordCloud
from termcolor import colored

# spacy for language processing
import spacy

# sklearn for feature extraction & modeling
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD, NMF
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib

# gensim for alternative models
```

```python
from gensim.models import LdaModel, LdaMulticore
from gensim.corpora import Dictionary
from gensim.matutils import Sparse2Corpus
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.6/site-
packages/scipy/sparse/sparsetools.py:21: DeprecationWarning:
`scipy.sparse.sparsetools` is deprecated!
scipy.sparse.sparsetools is a private module for scipy.sparse, and should not be
used.
  _deprecated()
```

[2]:
```python
%matplotlib inline
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (14.0, 8.7)
pyLDAvis.enable_notebook()
warnings.filterwarnings('ignore')
pd.options.display.float_format = '{:,.2f}'.format
```

1.2 Load BBC data

[3]:
```python
# change to your data path if necessary
DATA_DIR = Path('../data')
```

[4]:
```python
path = DATA_DIR / 'bbc'
files = path.glob('**/*.txt')
doc_list = []
for i, file in enumerate(files):
    with open(str(file), encoding='latin1') as f:
        topic = file.parts[-2]
        lines = f.readlines()
        heading = lines[0].strip()
        body = ' '.join([l.strip() for l in lines[1:]])
        doc_list.append([topic.capitalize(), heading, body])
```

1.2.1 Convert to DataFrame

[5]:
```python
docs = pd.DataFrame(doc_list, columns=['topic', 'heading', 'article'])
docs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 3 columns):
topic      2225 non-null object
heading    2225 non-null object
article    2225 non-null object
dtypes: object(3)
memory usage: 52.2+ KB
```

1.3 Create Train & Test Sets

[6]:
```
train_docs, test_docs = train_test_split(docs,
                                         stratify=docs.topic,
                                         test_size=50,
                                         random_state=42)
```

[7]:
```
train_docs.shape, test_docs.shape
```

[7]:
```
((2175, 3), (50, 3))
```

[8]:
```
pd.Series(test_docs.topic).value_counts()
```

[8]:
```
Sport            12
Business         11
Tech              9
Entertainment     9
Politics          9
Name: topic, dtype: int64
```

1.3.1 Vectorize train & test sets

[9]:
```
vectorizer = CountVectorizer(max_df=.2,
                             min_df=3,
                             stop_words='english',
                             max_features=2000)

train_dtm = vectorizer.fit_transform(train_docs.article)
words = vectorizer.get_feature_names()
train_dtm
```

[9]:
```
<2175x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 178572 stored elements in Compressed Sparse Row format>
```

[10]:
```
test_dtm = vectorizer.transform(test_docs.article)
test_dtm
```

[10]:
```
<50x2000 sparse matrix of type '<class 'numpy.int64'>'
        with 4160 stored elements in Compressed Sparse Row format>
```

1.4 LDA with gensim

1.4.1 Using CountVectorizer Input

[11]:
```
max_df = .2
min_df = 3
max_features = 2000
```

```
# used by sklearn: https://github.com/scikit-learn/scikit-learn/blob/master/
↪sklearn/feature_extraction/stop_words.py
stop_words = pd.read_csv('http://ir.dcs.gla.ac.uk/resources/linguistic_utils/
↪stop_words',
                         header=None,
                         squeeze=True).tolist()
```

[12]:
```
vectorizer = CountVectorizer(max_df=max_df,
                             min_df=min_df,
                             stop_words='english',
                             max_features=max_features)

train_dtm = vectorizer.fit_transform(train_docs.article)
test_dtm = vectorizer.transform(test_docs.article)
```

1.4.2 Convert sklearn DTM to gensim data structures

It faciltiates the conversion of DTM produced by sklearn to gensim data structures as follows:

[15]:
```
train_corpus = Sparse2Corpus(train_dtm, documents_columns=False)
test_corpus = Sparse2Corpus(test_dtm, documents_columns=False)
id2word = pd.Series(vectorizer.get_feature_names()).to_dict()
```

1.4.3 Train Model & Review Results

[20]:
```
LdaModel(corpus=train_corpus,
         num_topics=100,
         id2word=None,
         distributed=False,
         chunksize=2000,                  # Number of documents to be used in␣
↪each training chunk.
         passes=1,                        # Number of passes through the␣
↪corpus during training
         update_every=1,                  # Number of docs to be iterated␣
↪through for each update
         alpha='symmetric',
         eta=None,                        # a-priori belief on word probability
         decay=0.5,                       # percentage of previous lambda␣
↪forgotten when new document is examined
         offset=1.0,                      # controls slow down of the first␣
↪steps the first few iterations.
         eval_every=10,                   # estimate log perplexity
         iterations=50,                   # Maximum number of iterations␣
↪through the corpus
         gamma_threshold=0.001,           # Minimum change in the value of the␣
↪gamma parameters to continue iterating
```

```
        minimum_probability=0.01,         # Topics with a probability lower␣
↪than this threshold will be filtered out
        random_state=None,
        ns_conf=None,
        minimum_phi_value=0.01,           # if `per_word_topics` is True,␣
↪represents lower bound on term probabilities
        per_word_topics=False,            #  If True, compute a list of most␣
↪likely topics for each word with phi values multiplied by word count
        callbacks=None);
```

[16]:
```
num_topics = 5
topic_labels = ['Topic {}'.format(i) for i in range(1, num_topics+1)]
```

[17]:
```
lda_gensim = LdaModel(corpus=train_corpus,
                      num_topics=num_topics,
                      id2word=id2word)
```

[18]:
```
topics = lda_gensim.print_topics()
topics[0]
```

[18]:
```
(0,
 '0.008*"search" + 0.006*"net" + 0.006*"mail" + 0.005*"yahoo" + 0.005*"labour" +
0.005*"web" + 0.005*"tax" + 0.004*"says" + 0.004*"information" + 0.004*"oil"')
```

1.4.4 Evaluate Topic Coherence

Topic Coherence measures whether the words in a topic tend to co-occur together.

- It adds up a score for each distinct pair of top ranked words.
- The score is the log of the probability that a document containing at least one instance of the higher-ranked word also contains at least one instance of the lower-ranked word.

Large negative values indicate words that don't co-occur often; values closer to zero indicate that words tend to co-occur more often.

[21]:
```
coherence = lda_gensim.top_topics(corpus=train_corpus, coherence='u_mass')
```

Gensim permits topic coherence evaluation that produces the topic coherence and shows the most important words per topic:

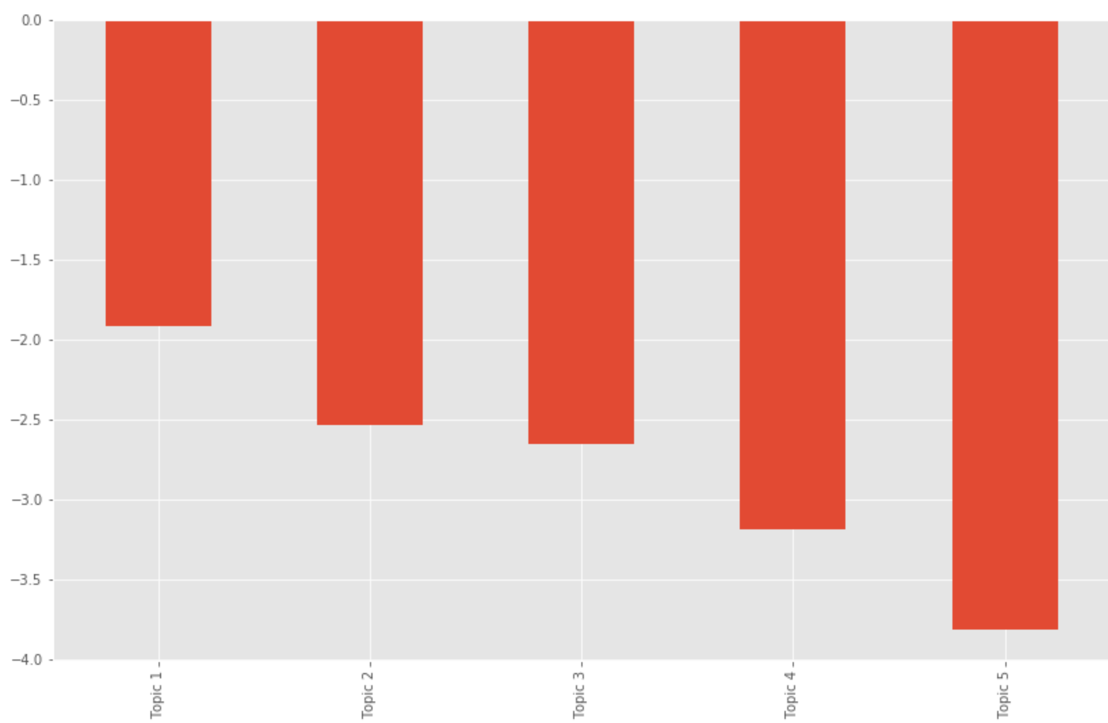
[22]:
```
topic_coherence = []
topic_words = pd.DataFrame()
for t in range(len(coherence)):
    label = topic_labels[t]
    topic_coherence.append(coherence[t][1])
    df = pd.DataFrame(coherence[t][0], columns=[(label, 'prob'), (label,␣
↪'term')])
```

```
    df[(label, 'prob')] = df[(label, 'prob')].apply(lambda x: '{:.2%}'.
↪format(x))
    topic_words = pd.concat([topic_words, df], axis=1)

topic_words.columns = pd.MultiIndex.from_tuples(topic_words.columns)
pd.set_option('expand_frame_repr', False)
topic_words.head().to_csv('topic_words.csv', index=False)
print(topic_words.head())

pd.Series(topic_coherence, index=topic_labels).plot.bar();
```

```
  Topic 1        Topic 2            Topic 3            Topic 4          Topic 5
     prob     term    prob      term    prob      term    prob     term    prob
term
0    0.70%   games   0.56%    united   0.97%    labour   0.78%  search   0.70%
digital
1    0.55%    game   0.52%        eu   0.81%     blair   0.62%     net   0.69%
wage
2    0.49%    2004   0.38%       aid   0.63%     party   0.59%    mail   0.60%
minimum
3    0.47%  market   0.38%  airlines   0.62%      film   0.51%   yahoo   0.58%
software
4    0.46%  prices   0.38%     state   0.53%  minister   0.51%  labour   0.55%
technology
```



1.4.5 Using gensim Dictionary

[23]:
```
docs = [d.split() for d in train_docs.article.tolist()]
docs = [[t for t in doc if t not in stop_words] for doc in docs]
```

[24]:
```
dictionary = Dictionary(docs)
dictionary.filter_extremes(no_below=min_df, no_above=max_df,␣
↪keep_n=max_features)
```

[25]:
```
corpus = [dictionary.doc2bow(doc) for doc in docs]
```

[26]:
```
print('Number of unique tokens: %d' % len(dictionary))
print('Number of documents: %d' % len(corpus))
```

```
Number of unique tokens: 2000
Number of documents: 2175
```

[27]:
```
num_topics = 5
chunksize = 500
passes = 20
iterations = 400
eval_every = None # Don't evaluate model perplexity, takes too much time.

temp = dictionary[0]  # This is only to "load" the dictionary.
id2word = dictionary.id2token
```

[30]:
```
model = LdaModel(corpus=corpus,
                 id2word=id2word,
                 chunksize=chunksize,
                 alpha='auto',
                 eta='auto',
                 iterations=iterations,
                 num_topics=num_topics,
                 passes=passes,
                 eval_every=eval_every)
```

[31]:
```
model.show_topics()
```

[31]:
```
[(0,
  '0.007*"company" + 0.007*"growth" + 0.006*"market" + 0.006*"economic" +
0.006*"oil" + 0.006*"sales" + 0.005*"firm" + 0.005*"rise" + 0.005*"economy" +
0.005*"prices"'),
 (1,
  '0.010*"technology" + 0.009*"mobile" + 0.008*"use" + 0.008*"digital" +
0.007*"music" + 0.007*"games" + 0.006*"users" + 0.006*"used" + 0.006*"software"
+ 0.006*"net"'),
 (2,
```

```
  '0.012*"Labour" + 0.011*"government" + 0.009*"Blair" + 0.007*"election" +
0.006*"public" + 0.006*"party" + 0.006*"Brown" + 0.005*"say" + 0.005*"Howard" +
0.005*"minister"'),
 (3,
  '0.009*"game" + 0.008*"win" + 0.008*"England" + 0.007*"good" + 0.006*"think" +
0.006*"play" + 0.005*"players" + 0.005*"got" + 0.005*"And" + 0.005*"it\'s"'),
 (4,
  '0.024*"best" + 0.021*"film" + 0.012*"won" + 0.009*"music" + 0.008*"British" +
0.008*"TV" + 0.007*"including" + 0.007*"director" + 0.007*"UK" + 0.007*"star"')]
```

1.4.6 Evaluating Topic Assignments on the Test Set

[32]:
```
docs_test = [d.split() for d in test_docs.article.tolist()]
docs_test = [[t for t in doc if t not in stop_words] for doc in docs_test]

test_dictionary = Dictionary(docs_test)
test_dictionary.filter_extremes(no_below=min_df, no_above=max_df,␣
 ↪keep_n=max_features)
test_corpus = [dictionary.doc2bow(doc) for doc in docs_test]
```

[33]:
```
gamma, _ = model.inference(test_corpus)
topic_scores = pd.DataFrame(gamma)
topic_scores.head(10)
```

[33]:

	0	1	2	3	4
0	0.11	0.07	0.09	2.81	67.32
1	6.82	60.50	27.93	0.10	0.05
2	0.11	32.94	0.09	51.46	6.79
3	61.13	0.07	32.06	0.10	0.05
4	0.11	0.07	0.09	115.79	4.33
5	63.55	0.07	32.64	0.10	0.05
6	42.69	0.07	0.09	2.51	0.05
7	0.11	0.07	26.56	22.62	0.05
8	103.20	0.07	26.73	0.10	6.29
9	54.08	0.07	0.09	0.10	7.07

[34]:
```
topic_probabilities = topic_scores.div(topic_scores.sum(axis=1), axis=0)
topic_probabilities.head()
```

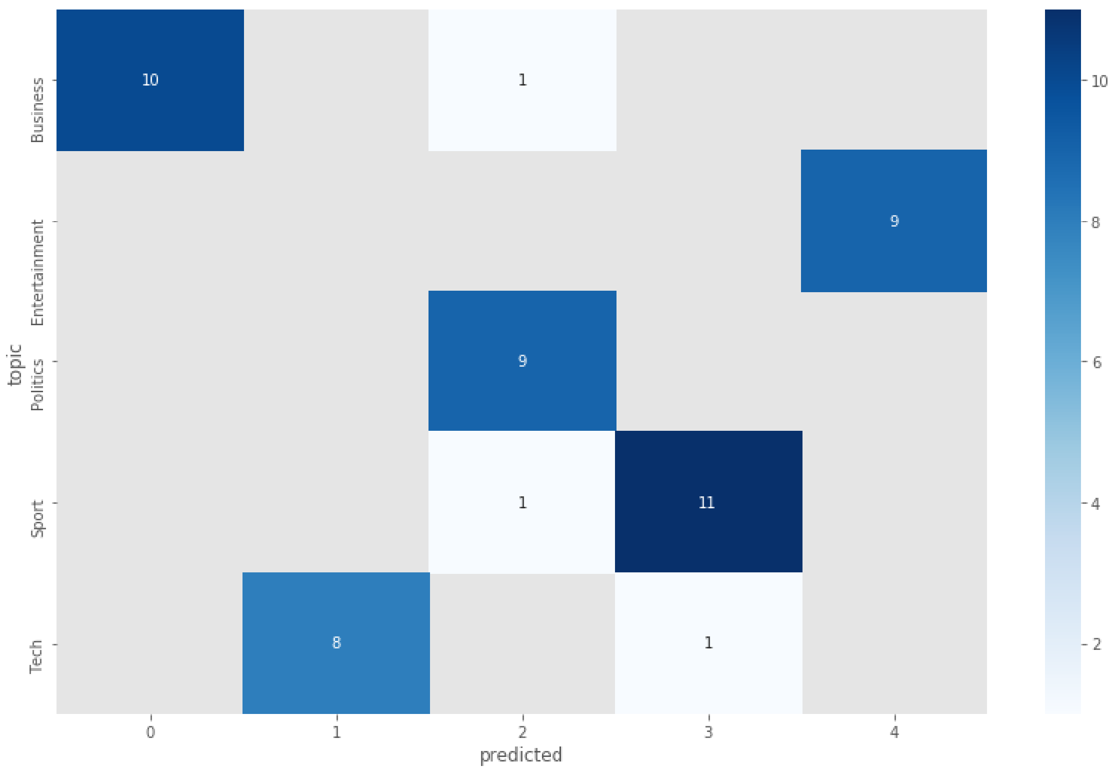
[34]:

	0	1	2	3	4
0	0.00	0.00	0.00	0.04	0.96
1	0.07	0.63	0.29	0.00	0.00
2	0.00	0.36	0.00	0.56	0.07
3	0.65	0.00	0.34	0.00	0.00
4	0.00	0.00	0.00	0.96	0.04

[35]:
```
topic_probabilities.idxmax(axis=1).head()
```

```
[35]: 0    4
      1    1
      2    3
      3    0
      4    3
      dtype: int64
```

```python
[36]: predictions = test_docs.topic.to_frame('topic').
      ↪assign(predicted=topic_probabilities.idxmax(axis=1).values)
      heatmap_data = predictions.groupby('topic').predicted.value_counts().unstack()
      sns.heatmap(heatmap_data, annot=True, cmap='Blues');
```



1.5 Resources

- pyLDAvis:
  - Talk by the Author and Paper by (original) Author
  - Documentation
- LDA:
  - David Blei Homepage @ Columbia
  - Introductory Paper and more technical review paper
  - Blei Lab @ GitHub
- Topic Coherence:
  - Exploring Topic Coherence over many models and many topics

– Paper on various Methods
– Blog Post - Overview