02_probabilistic_latent_analysis

September 29, 2021

1 Topic Modeling: probabilistic LSA / Non-negative Matrix Factorization

Probabilistic Latent Semantic Analysis (pLSA) takes a statistical perspective on LSA and creates a generative model to address the lack of theoretical underpinnings of LSA.

pLSA explicitly models the probability each co-occurrence of documents d and words w described by the DTM as a mixture of conditionally independent multinomial distributions that involve topics t. The symmetric formulation of this generative process of word-document co-occurrences assumes both words and documents are generated by the latent topic class, whereas the asymmetric model assumes the topics are selected given the document, and words result in a second step given the topic.

$$P(w,d) = \underbrace{\sum_t P(d \mid t) P(w \mid t)}_{\text{symmetric}} = \underbrace{P(d) \sum_t P(t \mid d) P(w \mid t)}_{\text{asymmetric}}$$

The number of topics is a hyperparameter chosen prior to training and is not learned from the data.

The benefits of using a probability model is that we can now compare models by evaluating the probability they assign to new documents given the parameters learned during training.

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
%matplotlib inline

from pathlib import Path
from random import randint
import numpy as np
import pandas as pd

# sklearn for feature extraction & modeling
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.model_selection import train_test_split
```

```python
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```python
np.random.seed(42)
sns.set_style('whitegrid')
pd.options.display.float_format = '{:,.2f}'.format
```

1.2 Load BBC data

[4]:
```python
# change to your data path if necessary
DATA_DIR = Path('../data')
```

[5]:
```python
path = DATA_DIR / 'bbc'
files = sorted(list(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

[6]:
```python
docs = pd.DataFrame(doc_list, columns=['Category', 'Heading', 'Article'])
docs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  2225 non-null   object
 1   Heading   2225 non-null   object
 2   Article   2225 non-null   object
dtypes: object(3)
memory usage: 52.3+ KB
```

1.3 Create Train & Test Sets

[7]:
```python
train_docs, test_docs = train_test_split(docs,
                                         stratify=docs.Category,
                                         test_size=50,
                                         random_state=42)
```

[8]:
```python
train_docs.shape, test_docs.shape
```

[8]:
```
((2175, 3), (50, 3))
```

[9]:
```python
pd.Series(test_docs.Category).value_counts()
```

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

1.3.1 Vectorize train & test sets

[10]:
```python
vectorizer = TfidfVectorizer(max_df=.2,
                             min_df=.01,
                             stop_words='english')

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

[10]:
```
<2175x2907 sparse matrix of type '<class 'numpy.float64'>'
        with 205235 stored elements in Compressed Sparse Row format>
```

[11]:
```python
test_dtm = vectorizer.transform(test_docs.Article)
test_dtm
```

[11]:
```
<50x2907 sparse matrix of type '<class 'numpy.float64'>'
        with 4649 stored elements in Compressed Sparse Row format>
```

1.3.2 Get token counts

[12]:
```python
train_token_count = train_dtm.sum(0).A.squeeze()
tokens = vectorizer.get_feature_names()
word_count = pd.Series(train_token_count, index=tokens).
 ↪sort_values(ascending=False)
word_count.head(10)
```

[12]:
```
film        41.47
game        36.54
best        36.33
labour      32.75
music       30.97
company     29.73
election    28.09
```

```
england     28.08
market      27.67
party       27.44
dtype: float64
```

1.4 probabilistic Latent Semantic Analysis

1.4.1 Implementation using Non-Negative Matrix Factorization

pLSI has been shown to be equivalent to Non-Negative Matrix Factorization with Kullback-Leibler Divergence objective.

pLSI is equivalent to Non-Negative Matrix Factorization using a Kullback-Leibler Divergence objective (see references on GitHub). Hence, we can use the sklearn.decomposition.NMF class to implement this model, following closely the LSA example.

[13]:
```
n_components = 5
topic_labels = ['Topic {}'.format(i) for i in range(1, n_components+1)]
```

Using the same train-test split of the DTM produced by the TfidfVectorizer, we fit pLSA like so:

[14]:
```
nmf = NMF(n_components=n_components,
          random_state=42,
          solver='mu',
          beta_loss='kullback-leibler',
          max_iter=1000)
nmf.fit(train_dtm)
```

[14]:
```
NMF(beta_loss='kullback-leibler', max_iter=1000, n_components=5,
    random_state=42, solver='mu')
```

We get a measure of the reconstruction error that is a substitute for the explained variance measure for LSI:

[15]:
```
nmf.reconstruction_err_
```

[15]:
```
315.93734706741594
```

1.4.2 Explore Topics

[16]:
```
train_doc_topics = nmf.transform(train_dtm)
train_doc_topics.shape
```

[16]:
```
(2175, 5)
```

[17]:
```
i = randint(0, len(train_docs))
(train_docs.iloc[i, :2].append(pd.Series(train_doc_topics[i],
                                         index=topic_labels)))
```

[17]:

```
Category                                Politics
Heading     UK firms 'embracing e-commerce'
Topic 1                                     0.04
Topic 2                                     0.01
Topic 3                                     0.00
Topic 4                                     0.00
Topic 5                                     0.06
dtype: object
```

[18]:

```
train_result = pd.DataFrame(data=train_doc_topics,
                            columns=topic_labels,
                            index=train_docs.Category)
```

[19]:

```
test_eval = pd.DataFrame(data=nmf.transform(test_dtm),
                         columns=topic_labels,
                         index=test_docs.Category)
```

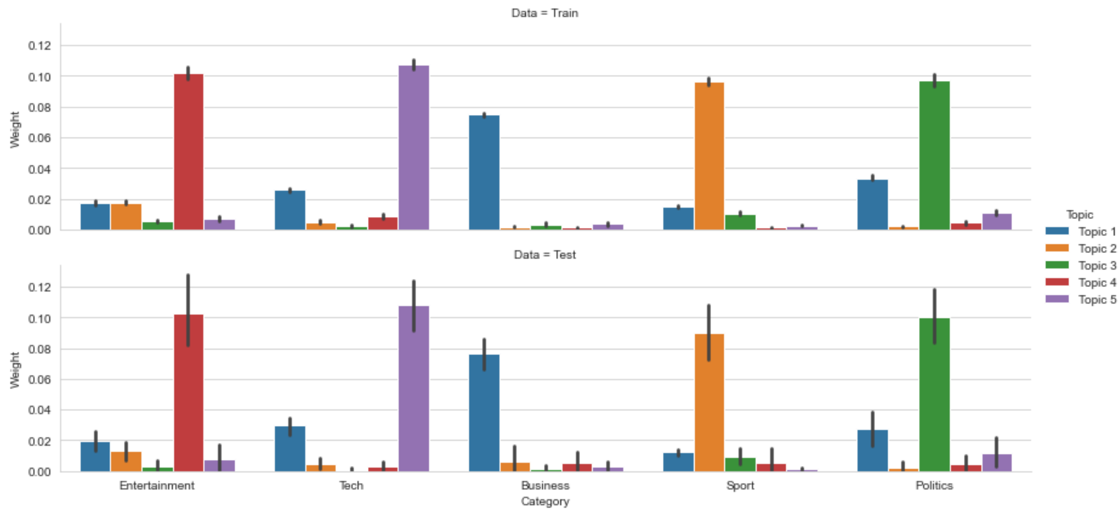
Due to its probabilistic nature, pLSA produces only positive topic weights that result in more straightforward topic-category relationships for the test and training sets:

[20]:

```
result = pd.melt(train_result.assign(Data='Train')
                 .append(test_eval.assign(Data='Test'))
                 .reset_index(),
                 id_vars=['Data', 'Category'],
                 var_name='Topic',
                 value_name='Weight')

result = pd.melt(train_result.assign(Data='Train')
                 .append(test_eval.assign(Data='Test'))
                 .reset_index(),
                 id_vars=['Data', 'Category'],
                 var_name='Topic',
                 value_name='Weight')

g =sns.catplot(x='Category',
               y='Weight',
               hue='Topic',
               row='Data',
               kind='bar',
               data=result,
               height=3,
               aspect=4);
```



1.4.3 Most important words by topic

We can also see that the word lists that describe each topic begin to make more sense, e.g. the 'Entertainment' category is most directly associated with Topic 4 that includes the words 'film', 'start', etc.

[21]:
```
topics = pd.DataFrame(nmf.components_.T,
                      index=tokens,
                      columns=topic_labels)
topics.loc[word_count.head(10).index]
```

[21]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
film	0.01	0.00	0.00	0.95	0.00
game	0.00	0.53	0.00	0.00	0.12
best	0.05	0.15	0.02	0.53	0.00
labour	0.02	0.00	0.67	0.00	0.00
music	0.00	0.00	0.00	0.37	0.30
company	0.38	0.00	0.00	0.00	0.00
election	0.03	0.00	0.56	0.00	0.00
england	0.04	0.38	0.07	0.00	0.00
market	0.35	0.00	0.00	0.00	0.00
party	0.01	0.00	0.55	0.01	0.02

[22]:
```
fig, ax = plt.subplots(figsize=(12, 4))
top_words, top_vals = pd.DataFrame(), pd.DataFrame()
for topic, words_ in topics.items():
    top10 = words_.nlargest(10).index
    vals = words_.loc[top10].values
    top_vals[topic] = vals
    top_words[topic] = top10.tolist()
```

```python
sns.heatmap(pd.DataFrame(top_vals),
            annot=top_words,
            fmt = '',
            center=0,
            cmap=sns.diverging_palette(0, 255, sep=1, n=256),
            ax=ax);
ax.set_title('Top Words per Topic')
fig.tight_layout();
```

[23]:
```python
topics = pd.DataFrame(nmf.components_.T,
                      index=words,
                      columns=topic_labels)
topics.head()
```

[23]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
000m	0.00	0.07	0.00	0.00	0.00
10	0.18	0.07	0.00	0.08	0.00
100	0.10	0.00	0.00	0.00	0.04
100m	0.02	0.05	0.00	0.00	0.00
11	0.10	0.02	0.00	0.07	0.00

[24]:
```python
top_words = {}
for topic, words_ in topics.items():
    top_words[topic] = words_.nlargest(10).index.tolist()
pd.DataFrame(top_words)
```

[24]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
0	company	game	labour	film	mobile
1	market	win	election	best	technology
2	2004	england	blair	awards	software
3	firm	play	party	music	users
4	sales	club	brown	award	digital

5	european	team	minister	band	use
6	growth	match	howard	album	music
7	economy	cup	prime	films	computer
8	expected	players	britain	star	phone
9	group	old	chancellor	actor	games