04_lda_with_sklearn

September 29, 2021

0.1 Topic Modeling: Latent Dirichlet Allocation with sklearn

0.1.1 Imports & Settings

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

[2]:
```
%matplotlib inline

from collections import OrderedDict
from pathlib import Path

import numpy as np
import pandas as pd

# Visualization
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

# sklearn for feature extraction & modeling
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.model_selection import train_test_split
import joblib
```

[3]:
```
sns.set_style('white')
plt.rcParams['figure.figsize'] = (14.0, 8.7)
```

[4]:
```
pyLDAvis.enable_notebook()
```

[5]:
```python
# change to your data path if necessary
DATA_DIR = Path('../data')
data_path = DATA_DIR / 'bbc'
```

[6]:
```python
results_path = Path('results')
model_path = Path('results', 'bbc')
if not model_path.exists():
    model_path.mkdir(exist_ok=True, parents=True)
```

0.2 Load BBC data

Using the BBC data as before, we use sklearn.decomposition.LatentDirichletAllocation to train an LDA model with five topics.

[7]:
```python
files = sorted(list(data_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])
```

0.2.1 Convert to DataFrame

[8]:
```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):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   topic    2225 non-null   object
 1   heading  2225 non-null   object
 2   article  2225 non-null   object
dtypes: object(3)
memory usage: 52.3+ KB
```

0.3 Create Train & Test Sets

[9]:
```python
train_docs, test_docs = train_test_split(docs,
                                         stratify=docs.topic,
                                         test_size=125,
                                         random_state=42)
```

[10]:
```
train_docs.shape, test_docs.shape
```

[10]:
```
((2100, 3), (125, 3))
```

[11]:
```
pd.Series(test_docs.topic).value_counts()
```

[11]:
```
Sport            29
Business         29
Politics         23
Tech             22
Entertainment    22
Name: topic, dtype: int64
```

0.3.1 Vectorize train & test sets

[12]:
```
# experiments with different settings results yields the following␣
 ↪hyperparameters (see issue 50)
vectorizer = TfidfVectorizer(max_df=.11,
                             min_df=.026,
                             stop_words='english')

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

[12]:
```
<2100x1097 sparse matrix of type '<class 'numpy.float64'>'
        with 113356 stored elements in Compressed Sparse Row format>
```

[13]:
```
test_dtm = vectorizer.transform(test_docs.article)
test_dtm
```

[13]:
```
<125x1097 sparse matrix of type '<class 'numpy.float64'>'
        with 6779 stored elements in Compressed Sparse Row format>
```

0.4 LDA with sklearn

[14]:
```
n_components = 5
topic_labels = [f'Topic {i}' for i in range(1, n_components+1)]
```

[15]:
```
lda_base = LatentDirichletAllocation(n_components=n_components,
                                     n_jobs=-1,
                                     learning_method='batch',
                                     max_iter=10)
lda_base.fit(train_dtm)
```

[15]:
```
LatentDirichletAllocation(n_components=5, n_jobs=-1)
```

0.4.1 Persist model

The model tracks the in-sample perplexity during training and stops iterating once this measure stops improving. We can persist and load the result as usual with sklearn objects:

[16]:
```
joblib.dump(lda_base, model_path / 'lda_10_iter.pkl')
```

[16]:
```
['results/bbc/lda_10_iter.pkl']
```

[17]:
```
lda_base = joblib.load(model_path / 'lda_10_iter.pkl')
lda_base
```

[17]:
```
LatentDirichletAllocation(n_components=5, n_jobs=-1)
```

0.5 Explore topics & word distributions

[18]:
```
# pseudo counts
topics_count = lda_base.components_
print(topics_count.shape)
topics_count[:5]
```

```
(5, 1097)
```

[18]:
```
array([[8.31468034, 6.82423569, 9.47292822, …, 1.62405735, 6.75742624,
        1.59182715],
       [0.21908957, 0.6953881 , 0.81608557, …, 0.40814018, 2.41744349,
        0.20047286],
       [1.08540207, 4.36071935, 3.83209219, …, 0.80952393, 6.2400053 ,
        5.32256921],
       [3.56473365, 2.70835622, 1.96359096, …, 5.37990662, 0.20629507,
        4.65551798],
       [1.35376149, 2.4864043 , 3.4180758 , …, 2.13436728, 0.28239299,
        4.4297076 ]])
```

[19]:
```
topics_prob = topics_count / topics_count.sum(axis=1).reshape(-1, 1)
topics = pd.DataFrame(topics_prob.T,
                      index=words,
                      columns=topic_labels)
topics.head()
```

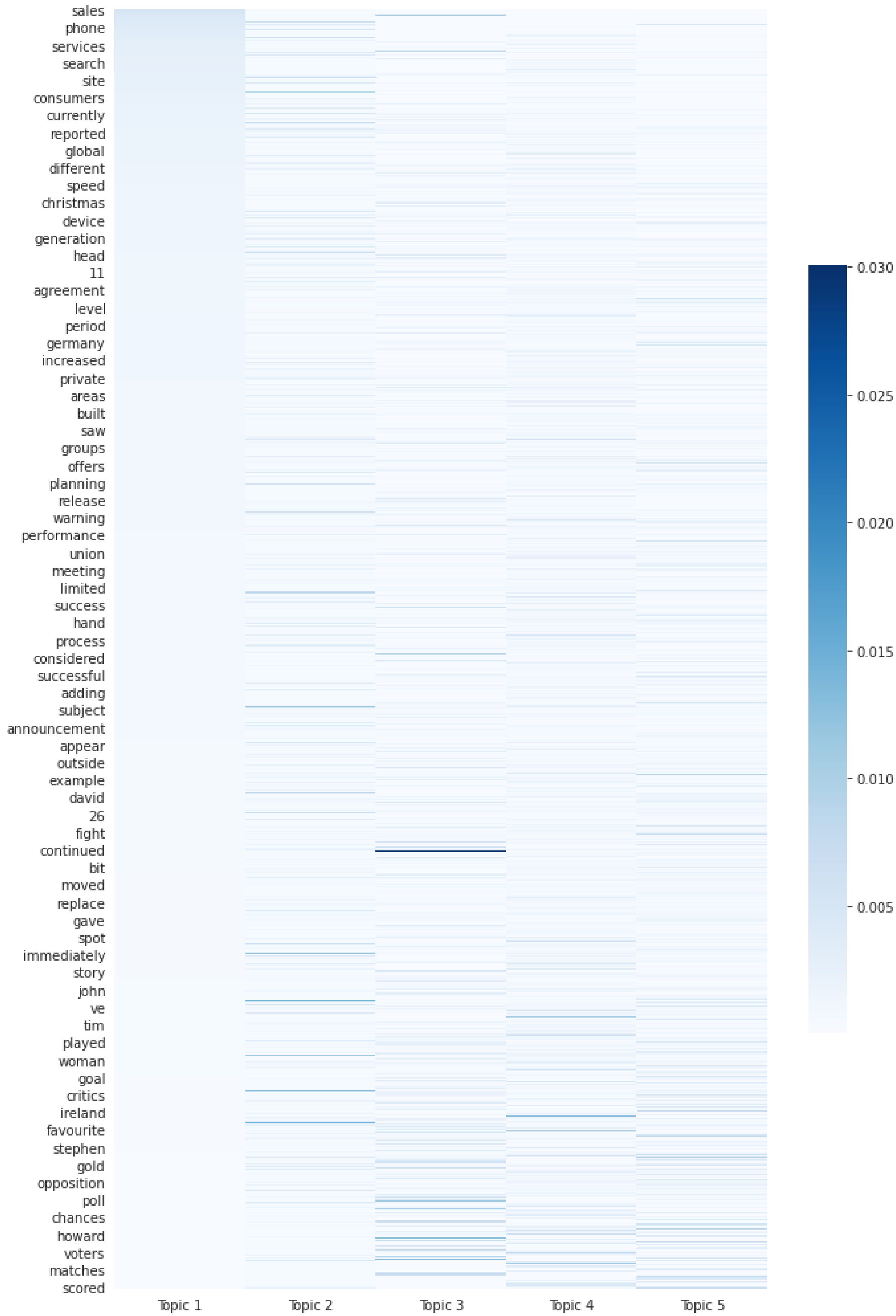
[19]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
100	0.001578	0.000320	0.000615	0.001148	0.000478
11	0.001295	0.001017	0.002473	0.000872	0.000879
12	0.001797	0.001194	0.002173	0.000632	0.001208
13	0.000899	0.000338	0.001881	0.000322	0.001292
14	0.000846	0.001080	0.001260	0.000731	0.001074

[20]:
```
# all words have positive probability for all topics
topics[topics.gt(0).all(1)].shape[0] == topics.shape[0]
```

[20]: True

[21]:
```
fig, ax = plt.subplots(figsize=(10, 14))
sns.heatmap(topics.sort_values(topic_labels, ascending=False),
            cmap='Blues', ax=ax, cbar_kws={'shrink': .6})
fig.tight_layout()
```



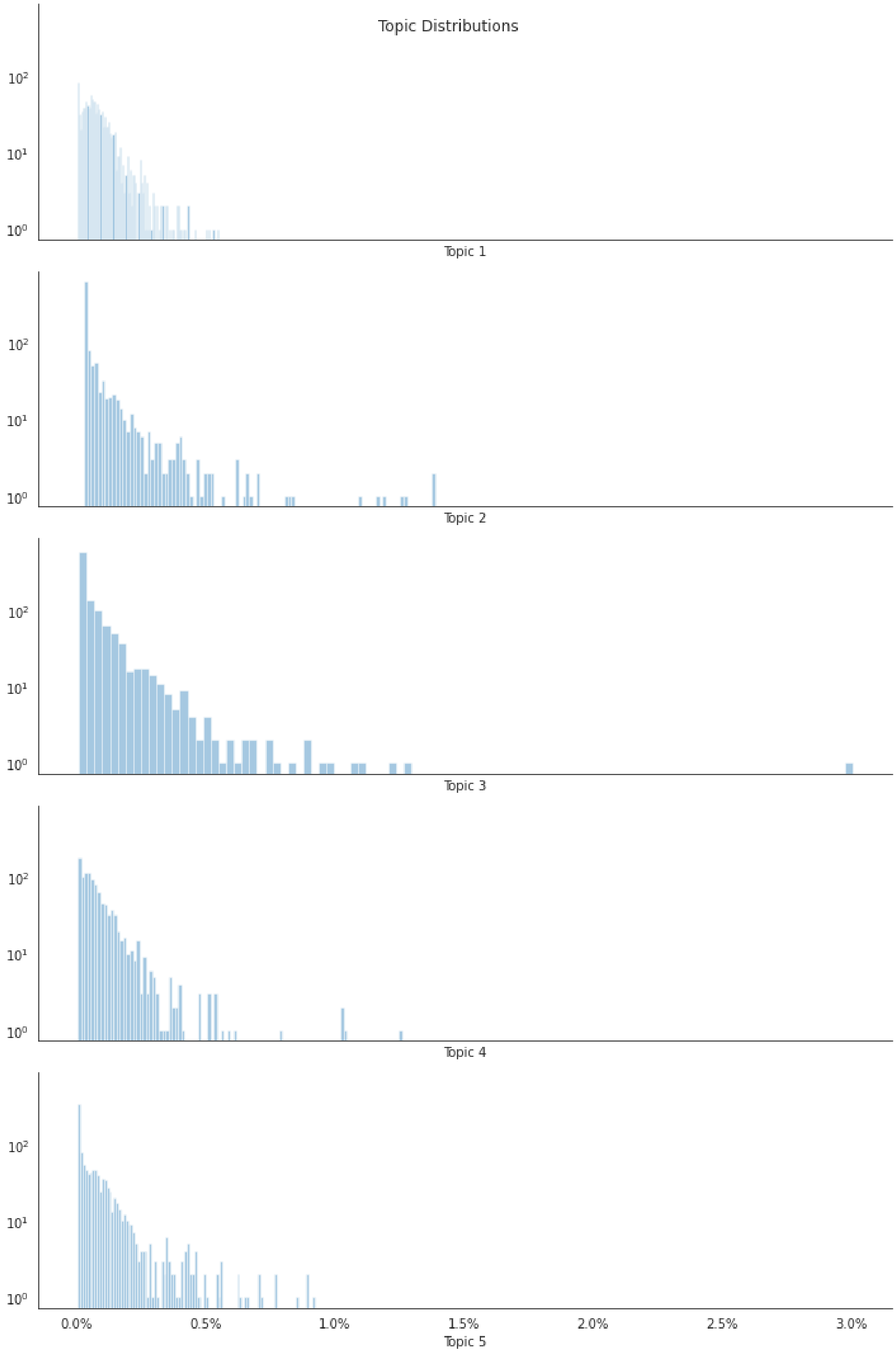
sales
phone
services
search
site
consumers
currently
reported
global
different
speed
christmas
device
generation
head
11
agreement
level
period
germany
increased
private
areas
built
saw
groups
offers
planning
release
warning
performance
union
meeting
limited
success
hand
process
considered
successful
adding
subject
announcement
appear
outside
example
david
26
fight
continued
bit
moved
replace
gave
spot
immediately
story
john
ve
tim
played
woman
goal
critics
ireland
favourite
stephen
gold
opposition
poll
chances
howard
voters
matches
scored
Topic 1
Topic 2
Topic 3
Topic 4
Topic 5
0.030
0.025
0.020
0.015
0.010
0.005

[22]:
```python
top_words = {}
for topic, words_ in topics.items():
    top_words[topic] = words_.nlargest(10).index.tolist()
pd.DataFrame(top_words)
```

[22]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
0	sales	charges	film	labour	club
1	technology	drugs	awards	blair	team
2	growth	russian	award	election	match
3	mobile	bid	band	party	cup
4	software	court	star	brown	injury
5	music	women	music	howard	season
6	users	trial	album	prime	ireland
7	digital	offer	actor	eu	final
8	economy	case	films	secretary	chelsea
9	computer	russia	singer	lord	wales

[23]:
```python
fig, axes = plt.subplots(nrows=5, sharey=True, sharex=True, figsize=(10, 15))
for i, (topic, prob) in enumerate(topics.items()):
    sns.distplot(prob, ax=axes[i], bins=100, kde=False, norm_hist=False)
    axes[i].set_yscale('log')
    axes[i].xaxis.set_major_formatter(FuncFormatter(lambda x, _: '{:.1%}'.
 ↪format(x)))
fig.suptitle('Topic Distributions')
sns.despine()
fig.tight_layout()
```



Topic Distributions
10⁰ 10¹ 10²
Topic 1
Topic 2
Topic 3
Topic 4
0.0% 0.5% 1.0% 1.5% 2.0% 2.5% 3.0%
Topic 5

0.6 Evaluate Fit on Train Set

[24]:
```
train_preds = lda_base.transform(train_dtm)
train_preds.shape
```

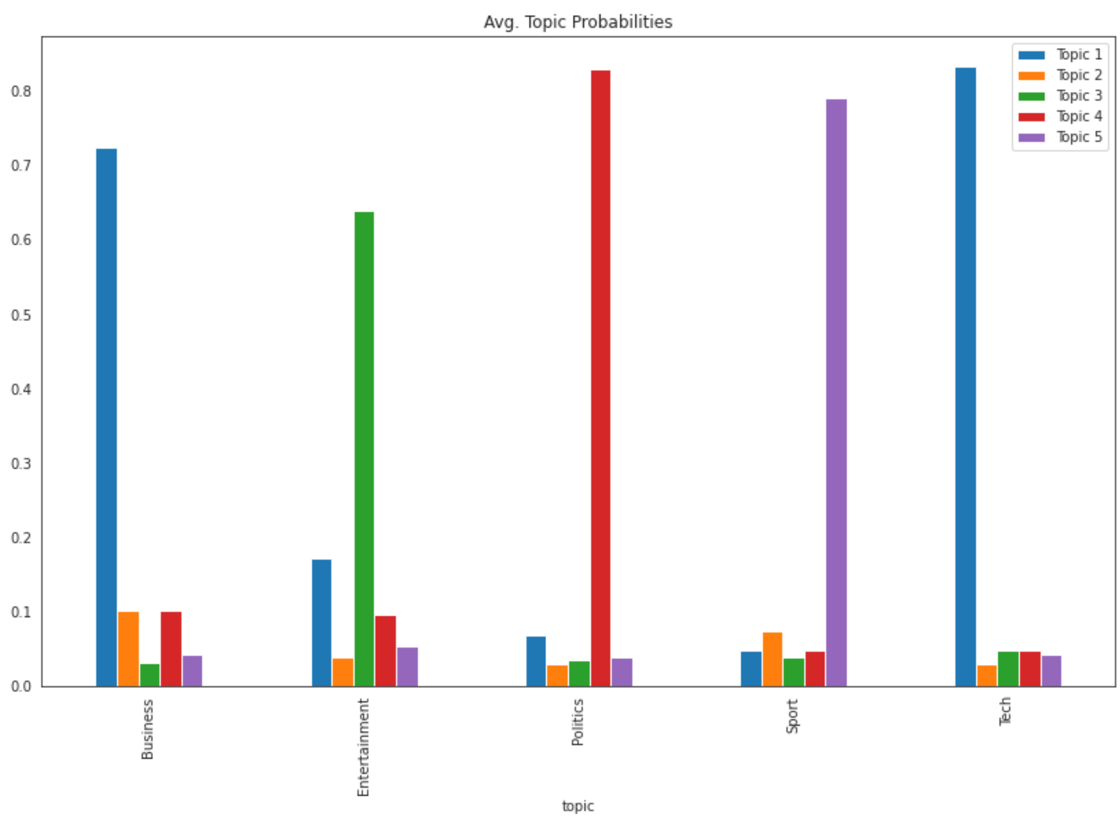
[24]:
```
(2100, 5)
```

[25]:
```
train_eval = pd.DataFrame(train_preds, columns=topic_labels, index=train_docs.
↪topic)
train_eval.head()
```

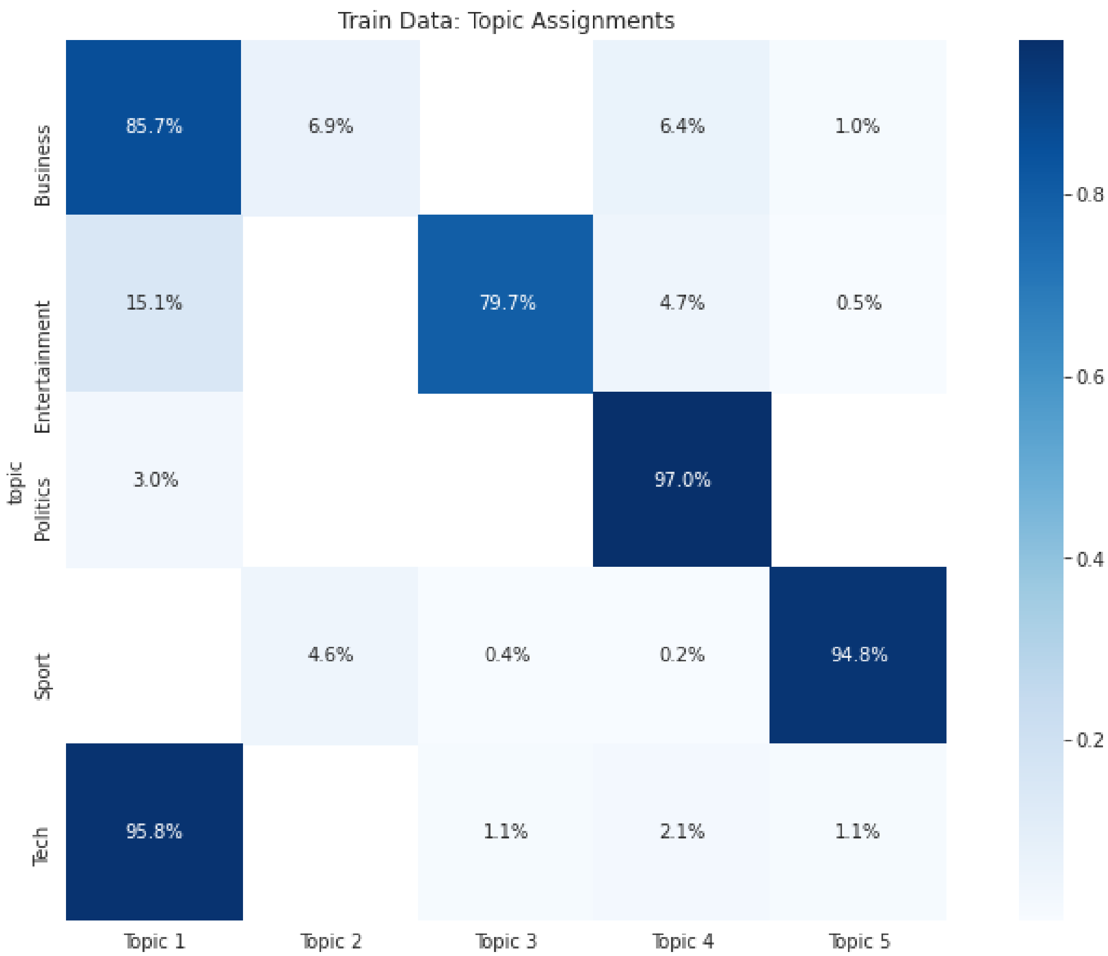
[25]:

topic	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
Sport	0.029769	0.029407	0.029715	0.029823	0.881286
Tech	0.875581	0.030722	0.032709	0.030722	0.030266
Politics	0.021231	0.021085	0.021013	0.915498	0.021173
Sport	0.041099	0.040811	0.041030	0.041172	0.835889
Tech	0.809909	0.029494	0.042780	0.029660	0.088158

[26]:
```
train_eval.groupby(level='topic').mean().plot.bar(title='Avg. Topic␣
↪Probabilities');
```



[27]:
```
df = train_eval.groupby(level='topic').idxmax(
    axis=1).reset_index(-1, drop=True)
sns.heatmap(df.groupby(level='topic').value_counts(normalize=True)
            .unstack(-1), annot=True, fmt='.1%', cmap='Blues', square=True)
plt.title('Train Data: Topic Assignments');
```



0.7 Evaluate Fit on Test Set

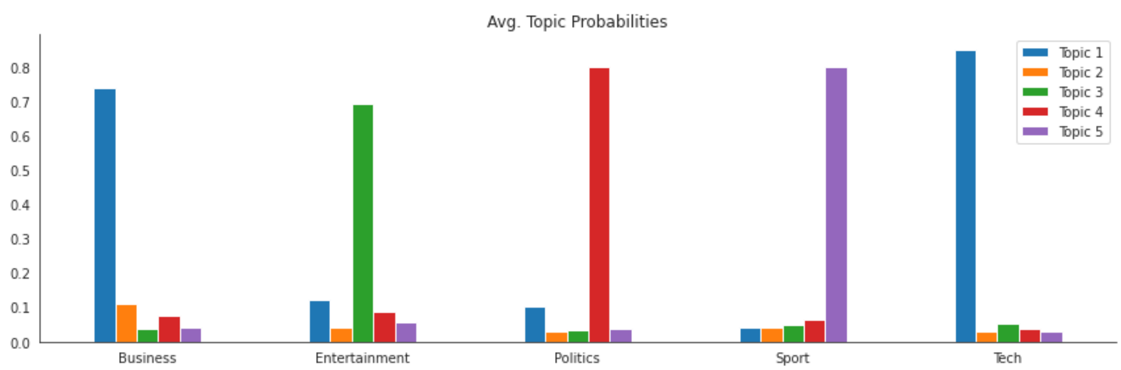
[28]:
```
test_preds = lda_base.transform(test_dtm)
test_eval = pd.DataFrame(test_preds, columns=topic_labels, index=test_docs.
 ↪topic)
test_eval.head()
```

[28]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
topic					
Politics	0.025285	0.025023	0.025241	0.897775	0.026675
Sport	0.037109	0.215594	0.036951	0.036369	0.673977
Business	0.842231	0.031315	0.032697	0.031516	0.062242
Business	0.875120	0.031362	0.031252	0.031147	0.031118
Tech	0.880013	0.029674	0.030008	0.030047	0.030258

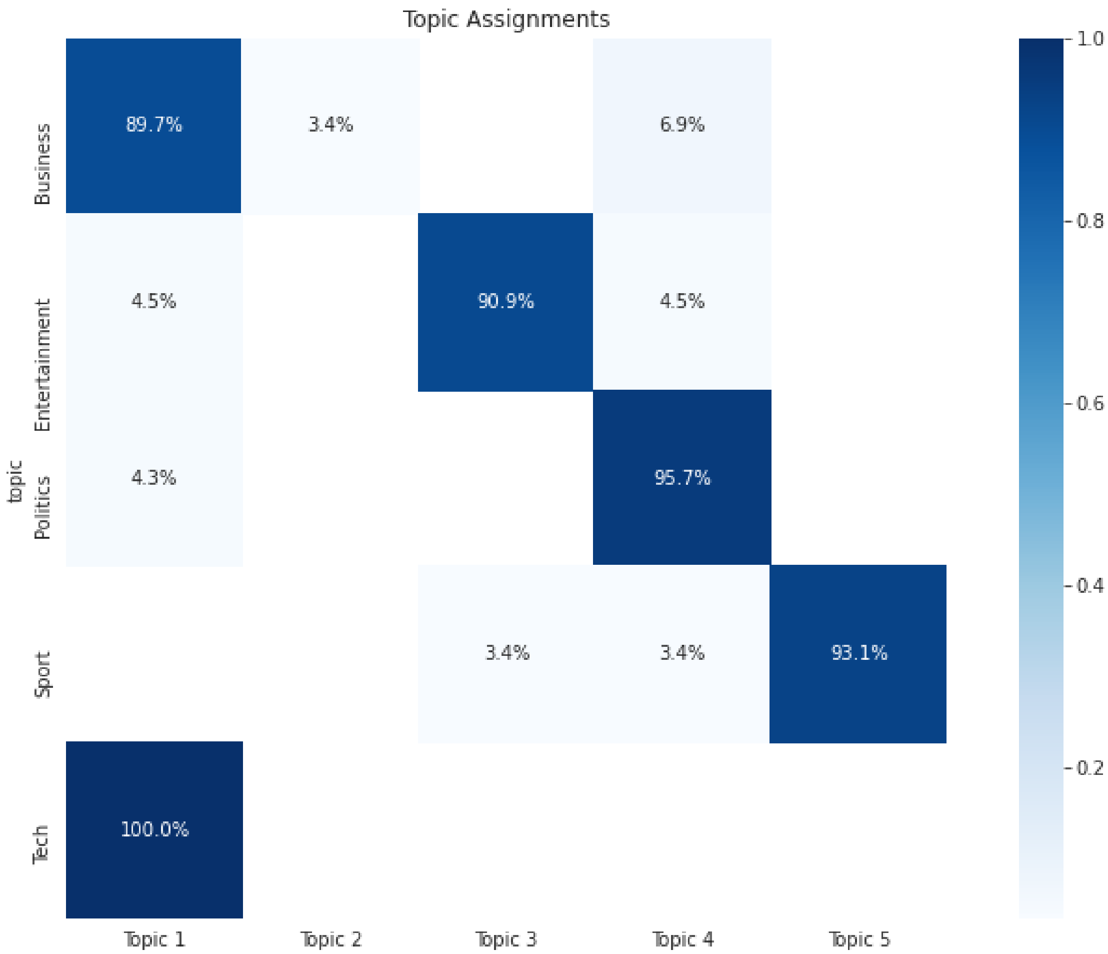
[29]:

```
test_eval.groupby(level='topic').mean().plot.bar(title='Avg. Topic␣
↪Probabilities',
                                                 figsize=(12, 4),
                                                 rot=0)
plt.xlabel('')
sns.despine()
plt.tight_layout()
```



[30]:

```
df = test_eval.groupby(level='topic').idxmax(axis=1).reset_index(-1, drop=True)
sns.heatmap(df.groupby(level='topic').value_counts(normalize=True).unstack(-1),
            annot=True, fmt='.1%', cmap='Blues', square=True)
plt.title('Topic Assignments');
```



0.8 Retrain until perplexity no longer decreases

[31]:
```
lda_opt = LatentDirichletAllocation(n_components=5,
                                    n_jobs=-1,
                                    max_iter=500,
                                    learning_method='batch',
                                    evaluate_every=5,
                                    verbose=1,
                                    random_state=42)
lda_opt.fit(train_dtm)
```

```
iteration: 1 of max_iter: 500
iteration: 2 of max_iter: 500
iteration: 3 of max_iter: 500
iteration: 4 of max_iter: 500
iteration: 5 of max_iter: 500, perplexity: 1934.2397
iteration: 6 of max_iter: 500
```

```
iteration: 7 of max_iter: 500
iteration: 8 of max_iter: 500
iteration: 9 of max_iter: 500
iteration: 10 of max_iter: 500, perplexity: 1874.2381
iteration: 11 of max_iter: 500
iteration: 12 of max_iter: 500
iteration: 13 of max_iter: 500
iteration: 14 of max_iter: 500
iteration: 15 of max_iter: 500, perplexity: 1849.3832
iteration: 16 of max_iter: 500
iteration: 17 of max_iter: 500
iteration: 18 of max_iter: 500
iteration: 19 of max_iter: 500
iteration: 20 of max_iter: 500, perplexity: 1837.3395
iteration: 21 of max_iter: 500
iteration: 22 of max_iter: 500
iteration: 23 of max_iter: 500
iteration: 24 of max_iter: 500
iteration: 25 of max_iter: 500, perplexity: 1831.0049
iteration: 26 of max_iter: 500
iteration: 27 of max_iter: 500
iteration: 28 of max_iter: 500
iteration: 29 of max_iter: 500
iteration: 30 of max_iter: 500, perplexity: 1830.5178
iteration: 31 of max_iter: 500
iteration: 32 of max_iter: 500
iteration: 33 of max_iter: 500
iteration: 34 of max_iter: 500
iteration: 35 of max_iter: 500, perplexity: 1828.5662
iteration: 36 of max_iter: 500
iteration: 37 of max_iter: 500
iteration: 38 of max_iter: 500
iteration: 39 of max_iter: 500
iteration: 40 of max_iter: 500, perplexity: 1828.4966
```

[31]:
```
LatentDirichletAllocation(evaluate_every=5, max_iter=500, n_components=5,
                          n_jobs=-1, random_state=42, verbose=1)
```

[32]:
```python
joblib.dump(lda_opt, model_path / 'lda_opt.pkl')
```

[32]:
```
['results/bbc/lda_opt.pkl']
```

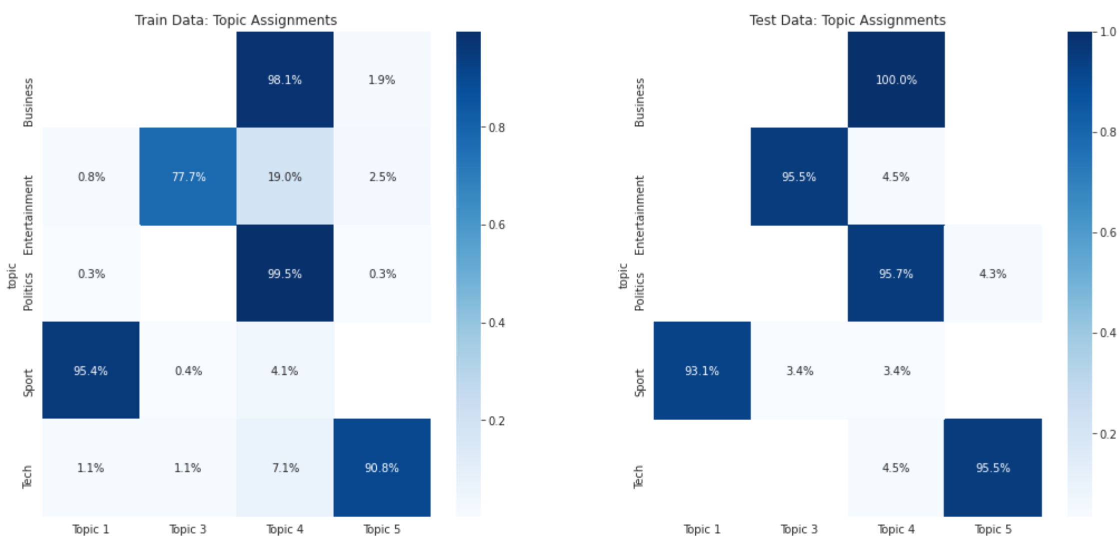
[33]:
```python
lda_opt = joblib.load(model_path / 'lda_opt.pkl')
```

[34]:
```python
train_opt_eval = pd.DataFrame(data=lda_opt.transform(train_dtm),
                              columns=topic_labels,
                              index=train_docs.topic)
```

[35]:
```
test_opt_eval = pd.DataFrame(data=lda_opt.transform(test_dtm),
                             columns=topic_labels,
                             index=test_docs.topic)
```

0.9 Compare Train & Test Topic Assignments

[36]:
```
fig, axes = plt.subplots(ncols=2, figsize=(18, 8))
source = ['Train', 'Test']
for i, df in enumerate([train_opt_eval, test_opt_eval]):
    df = df.groupby(level='topic').idxmax(axis=1).reset_index(-1, drop=True)
    sns.heatmap(df.groupby(level='topic').value_counts(normalize=True)
                .unstack(-1), annot=True, fmt='.1%', cmap='Blues', square=True,␣
 ↪ax=axes[i])
    axes[i].set_title('{} Data: Topic Assignments'.format(source[i]))
```



0.10 Explore misclassified articles

[48]:
```
test_assignments = test_docs.assign(predicted=test_opt_eval.idxmax(axis=1).
 ↪values)
test_assignments.head()
```

[48]:
```
         topic                           heading  \
1006  Politics    Kilroy launches 'Veritas' party
1358     Sport  Radcliffe eyes hard line on drugs
71    Business  S Korean consumers spending again
372   Business     Quiksilver moves for Rossignol
2151      Tech    Britons fed up with net service
```

```
                                               article predicted
1006   Ex-BBC chat show host and East Midlands MEP R…   Topic 4
1358   Paula Radcliffe has called for all athletes f…   Topic 1
71     South Korea looks set to sustain its revival …   Topic 4
372    Shares of Skis Rossignol, the world's largest…   Topic 4
2151   A survey conducted by PC Pro Magazine has rev…   Topic 5
```

[51]:
```python
misclassified = test_assignments[(test_assignments.topic == 'Entertainment') & (
    test_assignments.predicted == 'Topic 4')]
misclassified.heading
```

[51]:
```
677    Campaigners attack MTV 'sleaze'
Name: heading, dtype: object
```

[52]:
```python
misclassified.article.tolist()
```

[52]:
```
[' MTV has been criticised for "incessant sleaze" by television indecency
campaigners in the US.  The Parents Television Council (PTC), which monitors
violence and sex on TV, said the cable music channel offered the "cheapest form"
of programming. The group is at the forefront of a vociferous campaign to clean
up American television. But a spokeswoman for MTV said it was "unfair and
inaccurate" to single out MTV for criticism.  The PTC monitored MTV\'s output
for 171 hours from 20 March to 27 March 2004, during the channel\'s Spring Break
coverage. In its report - MTV Smut Peddlers: Targeting Kids with Sex, Drugs and
Alcohol - the PTC said it witnessed 3,056 flashes of nudity or sexual situations
and 2,881 verbal references to sex. Brent Bozell, PTC president and conservative
activist said: "MTV is blatantly selling raunchy sex to kids. "Compared to
broadcast television programmes aimed at adults, MTV\'s programming contains
substantially more sex, foul language and violence - and MTV\'s shows are aimed
at children as young as 12. "There\'s no question that TV influences the
attitudes and perceptions of young viewers, and MTV is deliberately marketing
its raunch to millions of innocent children."  The watchdog decided to look at
MTV\'s programmes after Janet Jackson\'s infamous "wardrobe malfunction" at last
year\'s Super Bowl. The breast-baring incident generated 500,000 complaints and
CBS - which is owned by the same parent company as MTV - was quick to apologise.
MTV spokeswoman Jeannie Kedas said the network follows the same standards as
broadcasters and reflects the culture and what its viewers are interested in.
"It\'s unfair and inaccurate to paint MTV with that brush of irresponsibility,"
she said. "We think it\'s underestimating young people\'s intellect and level of
sophistication." Ms Kedas also highlighted the fact MTV won an award in 2004 for
the Fight for Your Rights series that focused on issues such as sexual health
and tolerance.']
```

0.11 PyLDAVis

LDAvis helps you interpret LDA results by answer 3 questions:

1. What is the meaning of each topic?

2. How prevalent is each topic?
3. How do topics relate to each other?

Topic visualization facilitates the evaluation of topic quality using human judgment. pyLDAvis is a python port of LDAvis, developed in R and D3.js. We will introduce the key concepts; each LDA implementation notebook contains examples.

pyLDAvis displays the global relationships among topics while also facilitating their semantic evaluation by inspecting the terms most closely associated with each individual topic and, inversely, the topics associated with each term. It also addresses the challenge that terms that are frequent in a corpus tend to dominate the multinomial distribution over words that define a topic. LDAVis introduces the relevance r of term w to topic t to produce a flexible ranking of key terms using a weight parameter 0<= <=1.

With ϕ_{wt} as the model's probability estimate of observing the term w for topic t, and as the marginal probability of w in the corpus:

$$r(w,k|\lambda) = \lambda \log(\phi_{kw}) + (1-\lambda) \log \frac{\phi_{kw}}{p_w}$$

The first term measures the degree of association of term t with topic w, and the second term measures the lift or saliency, i.e., how much more likely the term is for the topic than in the corpus.

The tool allows the user to interactively change to adjust the relevance, which updates the ranking of terms. User studies have found that =0.6 produces the most plausible results.

0.12 Refit using all data

[53]:
```
vectorizer = CountVectorizer(max_df=.5,
                             min_df=5,
                             stop_words='english',
                             max_features=2000)
dtm = vectorizer.fit_transform(docs.article)
```

[54]:
```
lda_all = LatentDirichletAllocation(n_components=5,
                                    max_iter=500,
                                    learning_method='batch',
                                    evaluate_every=10,
                                    random_state=42,
                                    verbose=1)
lda_all.fit(dtm)
```

```
iteration: 1 of max_iter: 500
iteration: 2 of max_iter: 500
iteration: 3 of max_iter: 500
iteration: 4 of max_iter: 500
iteration: 5 of max_iter: 500
iteration: 6 of max_iter: 500
iteration: 7 of max_iter: 500
iteration: 8 of max_iter: 500
```

```
iteration: 9 of max_iter: 500
iteration: 10 of max_iter: 500, perplexity: 1036.6739
iteration: 11 of max_iter: 500
iteration: 12 of max_iter: 500
iteration: 13 of max_iter: 500
iteration: 14 of max_iter: 500
iteration: 15 of max_iter: 500
iteration: 16 of max_iter: 500
iteration: 17 of max_iter: 500
iteration: 18 of max_iter: 500
iteration: 19 of max_iter: 500
iteration: 20 of max_iter: 500, perplexity: 1034.0495
iteration: 21 of max_iter: 500
iteration: 22 of max_iter: 500
iteration: 23 of max_iter: 500
iteration: 24 of max_iter: 500
iteration: 25 of max_iter: 500
iteration: 26 of max_iter: 500
iteration: 27 of max_iter: 500
iteration: 28 of max_iter: 500
iteration: 29 of max_iter: 500
iteration: 30 of max_iter: 500, perplexity: 1033.2341
iteration: 31 of max_iter: 500
iteration: 32 of max_iter: 500
iteration: 33 of max_iter: 500
iteration: 34 of max_iter: 500
iteration: 35 of max_iter: 500
iteration: 36 of max_iter: 500
iteration: 37 of max_iter: 500
iteration: 38 of max_iter: 500
iteration: 39 of max_iter: 500
iteration: 40 of max_iter: 500, perplexity: 1033.1438
```

[54]:
```
LatentDirichletAllocation(evaluate_every=10, max_iter=500, n_components=5,
                          random_state=42, verbose=1)
```

[55]:
```python
joblib.dump(lda_all, model_path /'lda_all.pkl')
```

[55]:
```
['results/bbc/lda_all.pkl']
```

[56]:
```python
lda_all = joblib.load(model_path / 'lda_all.pkl')
```

Lambda

- $\lambda = \mathbf{0}$: how probable is a word to appear in a topic - words are ranked on lift P(word | topic) / P(word)
- $\lambda = \mathbf{1}$: how exclusive is a word to a topic - words are purely ranked on P(word | topic)

The ranking formula is $\lambda * P(\text{word}|\text{topic}) + (1-\lambda) * \text{lift}$

User studies suggest $\lambda = 0.6$ works for most people.

[57]:
```
prepare(lda_all, dtm, vectorizer)
```

[57]:
```
PreparedData(topic_coordinates=              x         y  topics  cluster
Freq
topic
4      0.069615  0.052014       1        1  30.791906
3      0.004624 -0.114806       2        1  22.890616
2      0.193820 -0.004663       3        1  20.458475
1     -0.108541  0.168193       4        1  15.589063
0     -0.159517 -0.100738       5        1  10.269940, topic_info=          Term
Freq        Total Category  logprob  loglift
1203         mr  2987.000000  2987.000000  Default  30.0000  30.0000
405     company   677.000000   677.000000  Default  29.0000  29.0000
1025     labour   770.000000   770.000000  Default  28.0000  28.0000
728        film   839.000000   839.000000  Default  27.0000  27.0000
1208      music   810.000000   810.000000  Default  26.0000  26.0000
…           …          …           …        …        …       …
283    business   123.119682   386.230263   Topic5  -5.4726   1.1327
1822       told   141.396865   904.836094   Topic5  -5.3341   0.4198
206        bank   115.958418   360.365538   Topic5  -5.5325   1.1421
1084     london   117.264384   454.835103   Topic5  -5.5213   0.9204
1228       news   115.733343   510.509978   Topic5  -5.5344   0.7918

[357 rows x 6 columns], token_table=      Topic      Freq     Term
term
0          1  0.310466      000
0          2  0.211167      000
0          3  0.016340      000
0          4  0.331834      000
0          5  0.130723      000
…      …        …       …
1992       4  0.231871    years
1992       5  0.079643    years
1993       4  0.989479      yen
1997       5  0.990858  yugansk
1998       5  0.996523    yukos

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

0.13 Topics as WordClouds

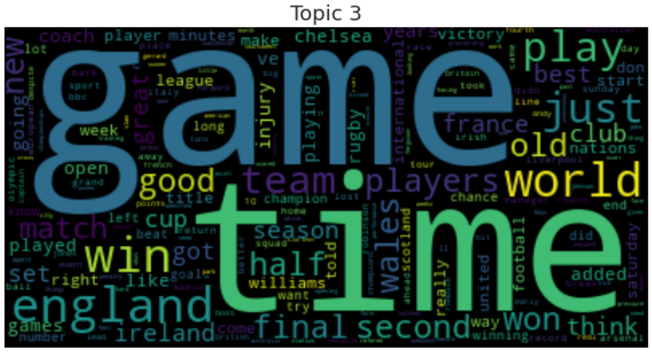
[58]:
```
topics_prob = lda_all.components_ / lda_all.components_.sum(axis=1).reshape(-1,␣
↪1)
topics = pd.DataFrame(topics_prob.T,
                      index=vectorizer.get_feature_names(),
                      columns=topic_labels)
```

[59]:
```
w = WordCloud()
fig, axes = plt.subplots(nrows=5, figsize=(15, 30))
axes = axes.flatten()
for t, (topic, freq) in enumerate(topics.items()):
    w.generate_from_frequencies(freq.to_dict())
    axes[t].imshow(w, interpolation='bilinear')
    axes[t].set_title(topic, fontsize=18)
    axes[t].axis('off')
```

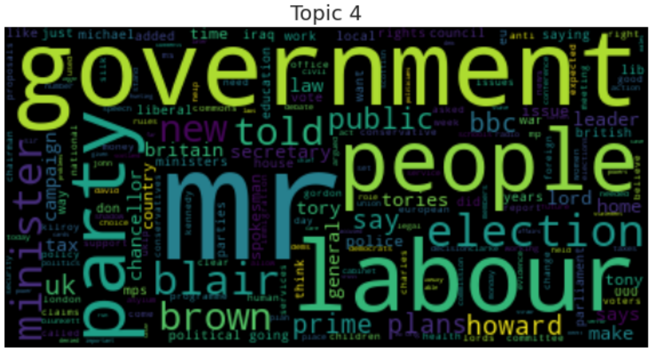
Topic 1

Topic 2

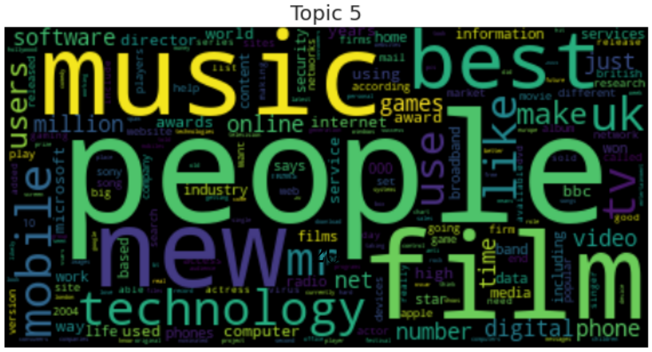
Topic 3



Topic 4



Topic 5



0.13.1 Visualize topic-word assocations per document

[60]:
```
dtm_ = pd.DataFrame(data=lda_all.transform(dtm),
                    columns=topic_labels,
                    index=docs.topic)
```

[61]:
```
dtm_.head()
```

[61]:
```
           Topic 1   Topic 2   Topic 3   Topic 4   Topic 5
topic
Business  0.217278  0.559808  0.001364  0.001375  0.220175
Business  0.001366  0.929723  0.001361  0.066198  0.001353
Business  0.992347  0.001911  0.001914  0.001910  0.001918
Business  0.001388  0.994458  0.001398  0.001375  0.001381
Business  0.566180  0.425551  0.002779  0.002776  0.002714
```

[62]:
```
color_dict = OrderedDict()
color_dict['Topic 1'] = {'color': 'white', 'on_color': 'on_blue'}
color_dict['Topic 2'] = {'color': 'white', 'on_color': 'on_green'}
color_dict['Topic 3'] = {'color': 'white', 'on_color': 'on_red'}
color_dict['Topic 4'] = {'color': 'white', 'on_color': 'on_magenta'}
color_dict['Topic 5'] = {'color': 'blue', 'on_color': 'on_yellow'}
```

[63]:
```
dtm_['article'] = docs.article.values
dtm_['heading'] = docs.heading.values
sample = dtm_[dtm_[topic_labels].gt(.05).all(1)]
sample
```

[63]:
```
                Topic 1   Topic 2   Topic 3   Topic 4   Topic 5  \
topic
Business       0.559255  0.222441  0.064607  0.055627  0.098070
Business       0.152645  0.519815  0.124709  0.119207  0.083624
Business       0.154587  0.419691  0.252344  0.075689  0.097689
Entertainment  0.215290  0.098103  0.150829  0.392351  0.143427
Entertainment  0.088833  0.179700  0.056604  0.449340  0.225522
Politics       0.222885  0.104102  0.092383  0.357693  0.222937
Tech           0.051874  0.173109  0.087102  0.113349  0.574566

                                                          article  \
topic
Business        One of Japan's best-known businessmen was arr…
Business        As the Aurora limped back to its dock on 20 J…
Business        Choking traffic jams in Beijing are prompting…
Entertainment   Proposals to open a museum dedicated to Jimi …
```

```
Entertainment   Musicians' groups are to tackle US visa regul…
Politics        A group of MPs and peers has called for a tig…
Tech            Dublin's hi-tech research laboratory, Media L…

                                          heading
topic
Business        Japanese mogul arrested for fraud
Business         Market unfazed by Aurora setback
Business        Beijingers fume over parking fees
Entertainment   Row threatens Hendrix museum plan
Entertainment     Musicians to tackle US red tape
Politics         Sport betting rules in spotlight
Tech             Dublin hi-tech labs to shut down
```

[64]:

```
colored_text = []
for word in sample.iloc[0, 5].split():
    try:
        topic = topics.loc[word.strip().lower()].idxmax()
        colored_text.append(colored(word, **color_dict[topic]))
    except:
        colored_text.append(word)

print(' '.join([colored(k, **v) for k, v in color_dict.items()]))
print('\n',sample.iloc[0, 6], '\n')
text = ' '.join(colored_text)
print(text)
```

```
Topic 1 Topic 2 Topic 3
Topic 4 Topic 5

 Japanese mogul arrested for fraud

One of Japan's best-known businessmen was arrested on
Thursday on charges of falsifying shareholder
information and selling shares
based on the false data. Yoshiaki Tsutsumi was once
ranked as the world's richest man and ran a
business spanning hotels, railways, construction and a baseball
team. His is the latest in a series of arrests of
top executives in Japan over business
scandals. He was taken away in a van
outside one of his Prince hotels in Tokyo. There was
a time when Mr Tsutsumi seemed untouchable.
Inheriting a large property business
from his father in the 1960s, he became one of Japan's most
powerful industrialists, with close
connections to many of the country's leading
```

politicians. He used his wealth and influence to
bring the Winter Olympic
Games to Nagano in 1998. But last year, he was
forced to resign from all the posts he held in his
business empire, after being accused of falsifying
the share-ownership structure of Seibu Railways, one of his companies. Under
Japanese stock market rules, no listed
company can be more than 80% owned by its
10 largest shareholders. Now Mr
Tsutsumi faces criminal charges and
the possibility of a prison sentence because he made it
look as if the 10 biggest
shareholders owned less than this amount. Seibu
Railways has been delisted from the stock exchange, its
share value has plunged and it is the
target of a takeover bid. Mr
Tsutsumi's fall from grace follows the arrests of
several other top executives in Japan as the
authorities try to curb the murky
business practices which were once widespread in
Japanese companies. His determination to stay at the
top at all costs may have had its roots in his childhood. The
illegitimate third son of a rich father, who made his
money buying up property as
Japan rebuilt after World War II, he
has described the demands his father
made. "I felt enormous pressure when I dined with
him and it was nothing but pain," Tsutsumi told a weekly
magazine in 1987. "He scolded me for pouring too much soy sauce or
told me fruit was not for children. He didn't let me
use the silk futon, saying it's a
luxury." There have been corporate governance issues
at some other Japanese companies too. Last year,
twelve managers from Mitsubishi Motors were charged with covering
up safety defects in their vehicles and three
executives from Japan's troubled UFJ bank were
charged with concealing the extent of the bank's bad
loans.