lda_yelp_reviews

September 29, 2021

1 Topic Modeling: Yelp Business Reviews

This notebook contains an example of LDA applied to six million business review on yelp.

1.1 Imports & Settings

[2]:
```
%matplotlib inline
import warnings
from collections import Counter
from pathlib import Path

import numpy as np
import pandas as pd
from scipy import sparse

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

# spacy for language processing
import spacy
from spacy.lang.en.stop_words import STOP_WORDS

# sklearn for feature extraction
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction import stop_words
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib

# gensim for topic models
from gensim.models import LdaModel
from gensim.models import CoherenceModel
from gensim.corpora import Dictionary
from gensim.matutils import Sparse2Corpus
```

```
# topic model viz
import pyLDAvis
from pyLDAvis.gensim import prepare
```

[3]:
```
plt.style.use('fivethirtyeight')
pyLDAvis.enable_notebook()
warnings.filterwarnings('ignore')
pd.options.display.float_format = '{:,.2f}'.format
```

[4]:
```
stop_words = set(pd.read_csv('http://ir.dcs.gla.ac.uk/resources/
 ↪linguistic_utils/stop_words',
                             header=None,
                             squeeze=True).tolist())
```

[5]:
```
experiment_path = Path('experiments')
data_path = Path('data')
clean_path = Path('data', 'clean_reviews.txt')
```

1.2 Load Yelp Reviews

[6]:
```
data_path = Path('..', '..', 'data', 'yelp')
```

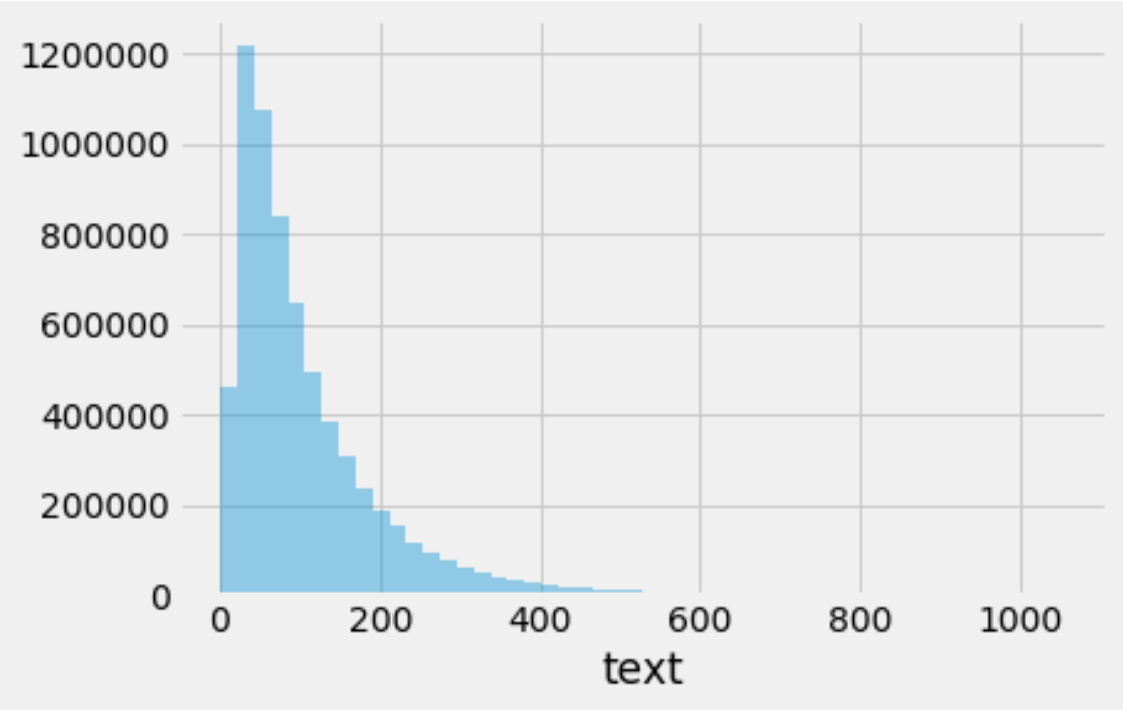
[7]:
```
reviews = pd.read_parquet(data_path / 'combined.parquet')
reviews.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6685900 entries, 0 to 6685899
Data columns (total 25 columns):
cool                  6685900 non-null int64
funny                 6685900 non-null int64
stars                 6685900 non-null float64
text                  6685900 non-null object
useful                6685900 non-null int64
year                  6685900 non-null int64
month                 6685900 non-null int64
average_stars         6685900 non-null float64
compliment_cool       6685900 non-null int64
compliment_cute       6685900 non-null int64
compliment_funny      6685900 non-null int64
compliment_hot        6685900 non-null int64
compliment_list       6685900 non-null int64
compliment_more       6685900 non-null int64
compliment_note       6685900 non-null int64
compliment_photos     6685900 non-null int64
compliment_plain      6685900 non-null int64
compliment_profile    6685900 non-null int64
compliment_writer     6685900 non-null int64
```

```
cool_user             6685900 non-null int64
fans                  6685900 non-null int64
funny_user            6685900 non-null int64
review_count          6685900 non-null int64
useful_user           6685900 non-null int64
member_yrs            6685900 non-null int64
dtypes: float64(2), int64(22), object(1)
memory usage: 1.3+ GB
```

1.2.1 Tokens per review

[8]:
```
sns.distplot(reviews.text.str.split().str.len(), kde=False);
```

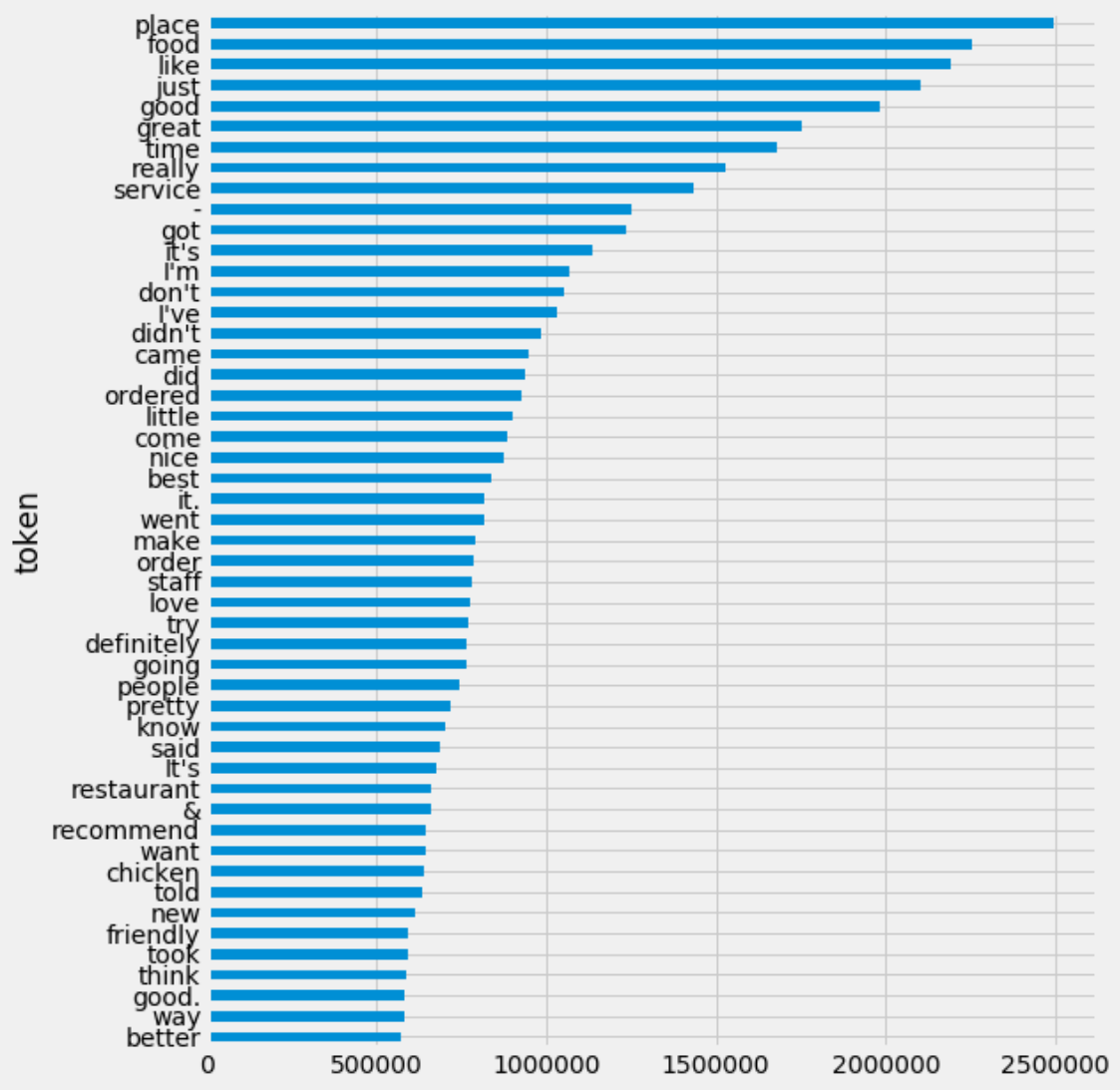


[9]:
```
token_count = Counter()
for i, doc in enumerate(reviews.text.tolist(), 1):
    if i % 1e6 == 0:
        print(i, end=' ', flush=True)
    token_count.update(doc.split())
```

```
1000000 2000000 3000000 4000000 5000000 6000000
```

[10]:
```
(pd.DataFrame(token_count.most_common(), columns=['token', 'count'])
 .pipe(lambda x: x[~x.token.str.lower().isin(stop_words)])
 .set_index('token')
 .squeeze()
```

```
.iloc[:50]
.sort_values()
.plot
.barh(figsize=(8, 10)));
```



1.2.2 Preprocessing

[12]:
```
def clean_doc(d):
    doc = []
    for t in d:
        if not any([t.is_stop, t.is_digit, not t.is_alpha, t.is_punct, t.
↪is_space, t.lemma_ == '-PRON-']):
            doc.append(t.lemma_)
```

```
    return ' '.join(doc)
```

[23]:
```
nlp = spacy.load('en')
nlp.max_length = 6000000
nlp.disable_pipes('ner')
```

[23]:
```
[('ner', <spacy.pipeline.EntityRecognizer at 0x7f5b8955fe08>)]
```

[24]:
```
nlp.pipe_names
```

[24]:
```
['tagger', 'parser']
```

[ ]:
```
iter_reviews = (review for review in reviews.text)
clean_reviews = []
for i, doc in enumerate(nlp.pipe(iter_reviews, batch_size=100, n_threads=8)):
    if i % 10000 == 0:
        print(f'{i/len(reviews):.2%}', end=' ', flush=True)
    clean_reviews.append(clean_doc(doc))
```

[ ]:
```
clean_reviews = [clean_doc(doc) for doc in parsed_reviews]
```

[ ]:
```
clean_path.write_text('\n'.join(clean_reviews))
```

1.3 Vectorize data

[6]:
```
docs = clean_path.read_text().split('\n')
len(docs)
```

[6]:
```
5990000
```

1.3.1 Explore cleaned data

[24]:
```
review_length, token_count = [], Counter()
for i, doc in enumerate(docs, 1):
    if i % 1e6 == 0:
        print(i, end=' ', flush=True)
    d = doc.split()
    review_length.append(len(d))
    token_count.update(d)
```

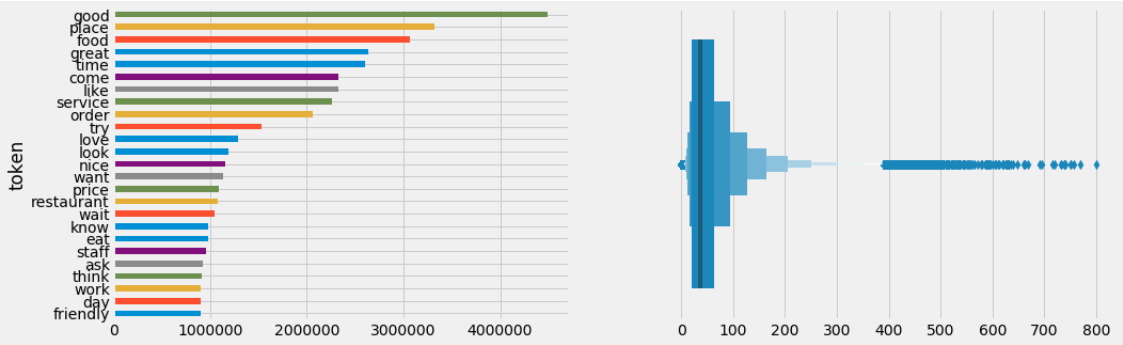
```
1000000 2000000 3000000 4000000 5000000
```

[43]:
```
fig, axes = plt.subplots(ncols=2, figsize=(15, 5))
(pd.DataFrame(token_count.most_common(), columns=['token', 'count'])
 .pipe(lambda x: x[~x.token.str.lower().isin(stop_words)])
 .set_index('token')
 .squeeze()
```

```
 .iloc[:25]
 .sort_values()
 .plot
 .barh(ax=axes[0]))

sns.boxenplot(x=pd.Series(review_length), ax=axes[1]);
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



[44]:
```
pd.Series(review_length).describe(percentiles=np.arange(.1, 1.0, .1))
```

[44]:
```
count   5,990,000.00
mean           50.88
std            46.50
min             0.00
10%            14.00
20%            18.00
30.0%          24.00
40%            29.00
50%            36.00
60%            45.00
70%            57.00
80%            74.00
90%           105.00
max           799.00
dtype: float64
```

[7]:
```
docs[:2]
```

[7]:
```
['the pizza okay not good prefer biaggio flamingo fort apache the chef much good
ny style pizza the pizzeria cosmo price quality lack personality food biaggio
good pick go italian family own home recipe people actually care like food not
```

```
pizzeria casino not care',
 'love place fiance and atleast week the portion huge food amazing love carne
asada great lunch special leticia super nice care think restaurant try cheese
enchilada sauce different and amazing']
```

[10]:
```
reviews.text.head(2)
```

[10]:
```
0    The pizza was okay. Not the best I've had. I p…
1    I love this place! My fiance And I go here atl…
Name: text, dtype: object
```

1.3.2 Set vocab parameters

[ ]:
```
min_df = 1000
max_df = .2
ngram_range = (1, 1)
binary = False
```

[319]:
```
vectorizer = CountVectorizer(stop_words='english',
                             min_df=min_df,
                             max_df=max_df,
                             ngram_range=ngram_range,
                             binary=binary)
dtm = doc_vect.fit_transform(docs)
tokens = doc_vect.get_feature_names()
dtm.shape
```

[321]:
```
corpus = Sparse2Corpus(dtm, documents_columns=False)
id2word = pd.Series(tokens).to_dict()
dictionary = Dictionary.from_corpus(corpus, id2word)
```

1.4 Train & Evaluate LDA Model

[45]:
```
def show_word_list(model, corpus, top=10, save=False):
    top_topics = model.top_topics(corpus=corpus, coherence='u_mass', topn=20)
    words, probs = [], []
    for top_topic, _ in top_topics:
        words.append([t[1] for t in top_topic[:top]])
        probs.append([t[0] for t in top_topic[:top]])

    fig, ax = plt.subplots(figsize=(model.num_topics*1.2, 5))
    sns.heatmap(pd.DataFrame(probs).T,
                annot=pd.DataFrame(words).T,
                fmt='',
                ax=ax,
                cmap='Blues',
```

```
                cbar=False)
    fig.tight_layout()
    if save:
        fig.savefig('yelp_wordlist', dpi=300)
```

[46]:
```
def show_coherence(model, corpus, tokens, top=10, cutoff=0.01):
    top_topics = model.top_topics(corpus=corpus, coherence='u_mass', topn=20)
    word_lists = pd.DataFrame(model.get_topics().T, index=tokens)
    order = []
    for w, word_list in word_lists.items():
        target = set(word_list.nlargest(top).index)
        for t, (top_topic, _) in enumerate(top_topics):
            if target == set([t[1] for t in top_topic[:top]]):
                order.append(t)

    fig, axes = plt.subplots(ncols=2, figsize=(15,5))
    title = f'# Words with Probability > {cutoff:.2%}'
    (word_lists.loc[:, order]>cutoff).sum().reset_index(drop=True).plot.
↪bar(title=title, ax=axes[1]);

    umass = model.top_topics(corpus=corpus, coherence='u_mass', topn=20)
    pd.Series([c[1] for c in umass]).plot.bar(title='Topic Coherence',␣
↪ax=axes[0])
    fig.tight_layout();
```

[47]:
```
def show_top_docs(model, corpus, docs):
    doc_topics = model.get_document_topics(corpus)
    df = pd.concat([pd.DataFrame(doc_topic,
                                 columns=['topicid', 'weight']).assign(doc=i)
                    for i, doc_topic in enumerate(doc_topics)])

    for topicid, data in df.groupby('topicid'):
        print(topicid, docs[int(data.sort_values('weight', ascending=False).
↪iloc[0].doc)])
        print(pd.DataFrame(lda.show_topic(topicid=topicid)))
```

[322]:
```
num_topics=25
chunksize=2000
passes=10
update_every=None
alpha='auto'
eta='auto'
decay=0.5
offset=1.0
eval_every=None
iterations=50
gamma_threshold=0.001
```

```
minimum_probability=0.01
minimum_phi_value=0.01
per_word_topics=False
```

[323]:
```
lda_model = LdaModel(corpus=doc_corpus,
                     id2word=doc_id2word,
                     num_topics=num_topics,
                     chunksize=chunksize,
                     update_every=update_every,
                     alpha=alpha,
                     eta=eta,
                     decay=decay,
                     offset=offset,
                     eval_every=eval_every,
                     passes=passes,
                     iterations=iterations,
                     gamma_threshold=gamma_threshold,
                     minimum_probability=minimum_probability,
                     minimum_phi_value=minimum_phi_value,
                     random_state=42)
```

```
CPU times: user 55.3 s, sys: 8.76 s, total: 1min 4s
Wall time: 51.7 s
```

[ ]:
```
2 ** (-lda_model.log_perplexity(exp_test_corpus))
```

We show results for one model using a vocabulary of 3,800 tokens based on min_df=0.1% and max_df=25% with a single pass to avoid length training time for 20 topics. We can use pyldavis topic_info attribute to compute relevance values for lambda=0.6 that produces the following word list

[ ]:
```
show_word_list(model=lda_model, corpus=exp_corpus)
```

[ ]:
```
show_coherence(model=lda_model, corpus=exp_corpus, tokens=exp_tokens)
```

[325]:
```
vis = prepare(doc_model, doc_corpus, doc_dictionary, mds='tsne')
pyLDAvis.display(vis)
```

[325]:
```
<IPython.core.display.HTML object>
```

[294]:
```
vis = prepare(doc_model, doc_corpus, doc_dictionary, mds='tsne')
pyLDAvis.display(vis)
```

[294]:
```
<IPython.core.display.HTML object>
```

1.5 Load Experiments

1.5.1 Load Document-Term Matrix

[48]:
```
max_df = .25    # [.1, .25, .5, 1.0]
min_df = .005   # [.001, .005, .01]
binary= False  # [True, False]
```

[49]:
```
vocab_path = experiment_path / str(min_df) / str(max_df) / str(int(binary))
exp_dtm = sparse.load_npz(vocab_path / f'dtm.npz')
exp_tokens = pd.read_csv(vocab_path / f'tokens.csv', header=None, squeeze=True)
exp_dtm.shape
```

[49]:
```
(5990000, 3787)
```

[50]:
```
exp_id2word = exp_tokens.to_dict()
exp_corpus = Sparse2Corpus(exp_dtm, documents_columns=False)
exp_dictionary = Dictionary.from_corpus(exp_corpus, exp_id2word)
```

[51]:
```
exp_train_dtm, exp_test_dtm = train_test_split(exp_dtm, test_size=.1)
exp_test_dtm
```

[51]:
```
<599000x3787 sparse matrix of type '<class 'numpy.int64'>'
        with 20453281 stored elements in Compressed Sparse Row format>
```

1.5.2 Set Model Parameters

[52]:
```
num_topics = 20 # [3, 5, 7, 10, 15, 20, 25, 50]
passes = 1    # [1]
```

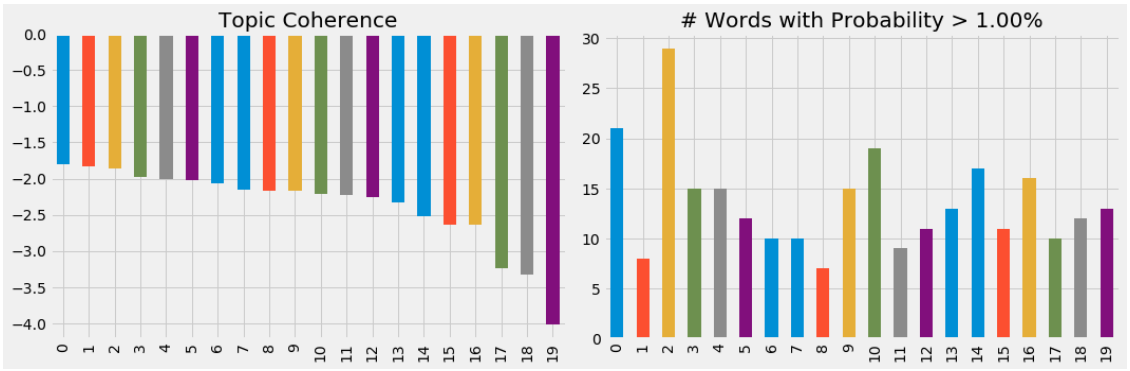
[53]:
```
exp_model_path = vocab_path / str(num_topics) / str(passes)
exp_lda = LdaModel.load(str(exp_model_path / 'lda'))
```

[ ]:
```
```

[54]:
```
show_word_list(model=exp_lda, corpus=exp_corpus, save=True)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	go	order	fry	car	price	coffee	recommend	bar	hair	order	store	staff	pizza	room	wait	year	location	de	review	tea
1	say	table	burger	call	pretty	breakfast	work	beer	nail	chicken	buy	feel	friendly	hotel	line	kid	dog	la	star	drink
2	know	wait	cheese	tell	sushi	cream	thank	drink	cut	rice	find	care	love	stay	hour	love	parking	et	yelp	water
3	tell	server	order	day	restaurant	ice	highly	menu	look	taco	shop	office	staff	vegas	long	fun	area	le	read	coupon
4	bad	ask	chicken	company	eat	chocolate	experience	night	salon	dish	price	massage	amazing	nice	get	old	park	pour	write	shake
5	ask	take	salad	phone	quality	cake	amazing	restaurant	color	soup	look	appointment	definitely	pool	minute	play	lot	un	birthday	green
6	want	minute	sauce	fix	think	try	need	wine	job	roll	item	need	this	strip	night	game	drive	con	mom	cup
7	get	waitress	sandwich	pay	bad	egg	job	nice	get	sauce	need	go	nice	floor	go	family	street	pas	daughter	milk
8	customer	drink	bread	charge	taste	flavor	guy	dinner	wash	try	selection	experience	super	night	open	class	live	que	wife	smoothie
9	not	restaurant	meat	say	bit	sweet	look	atmosphere	want	spicy	sell	patient	price	clean	day	go	this	du	sister	ice

[55]:
```
show_coherence(model=exp_lda, corpus=exp_corpus, tokens=exp_tokens)
```



[58]:
```
exp_vis = prepare(exp_lda, exp_corpus, exp_dictionary, mds='tsne')
```

[58]:
```
<IPython.core.display.HTML object>
```

[168]:
```
pyLDAvis.save_html(exp_vis, 'yelp_ldavis.html')
```

[71]:
```
pyLDAvis.display(exp_vis)
```

[71]:
```
<IPython.core.display.HTML object>
```

[170]:
```
terms = exp_vis.topic_info
terms = terms[terms.Category != 'Default']
terms['relevance'] = terms.logprob * .6 + terms.loglift * .4
```

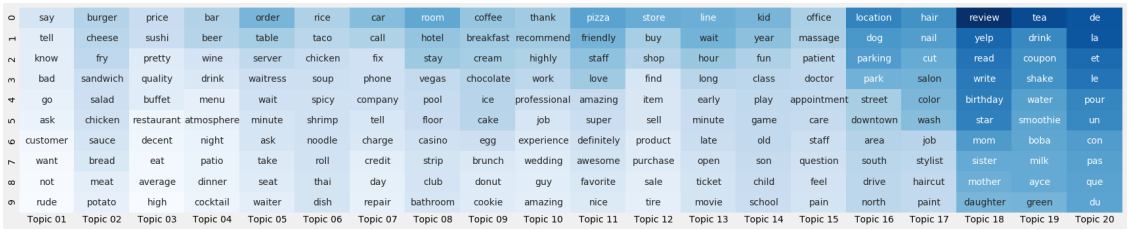
[108]:
```
top_by_relevance = (terms
                    .groupby('Category')
                    .apply(lambda x: x.nlargest(n=10, columns='relevance'))
                    .reset_index('term', drop=True)
                   .loc[:, ['Term', 'relevance']])
top_by_relevance.head()
```

[108]:

Category	Term	relevance
Topic1	say	-1.70
Topic1	tell	-1.89
Topic1	know	-1.90
Topic1	bad	-1.91
Topic1	go	-1.98

[156]:
```
relevance, terms = pd.DataFrame(), pd.DataFrame()
for topic, data in top_by_relevance.groupby(level='Category'):
    t = topic[:5] + f' {int(topic[5:]):0>2}'
    terms[t] = data.Term.tolist()
    relevance[t] = data.relevance.tolist()
```

[159]:
```
fig, ax = plt.subplots(figsize=(num_topics*1.2, 5))
sns.heatmap(relevance.sort_index(1),
            annot=terms.sort_index(1),
            fmt='',
            ax=ax,
            cmap='Blues',
            cbar=False)
fig.tight_layout()
fig.savefig('yelp_review_wordlist', dpi=300)
```



1.6 LDAMultiCore Timing

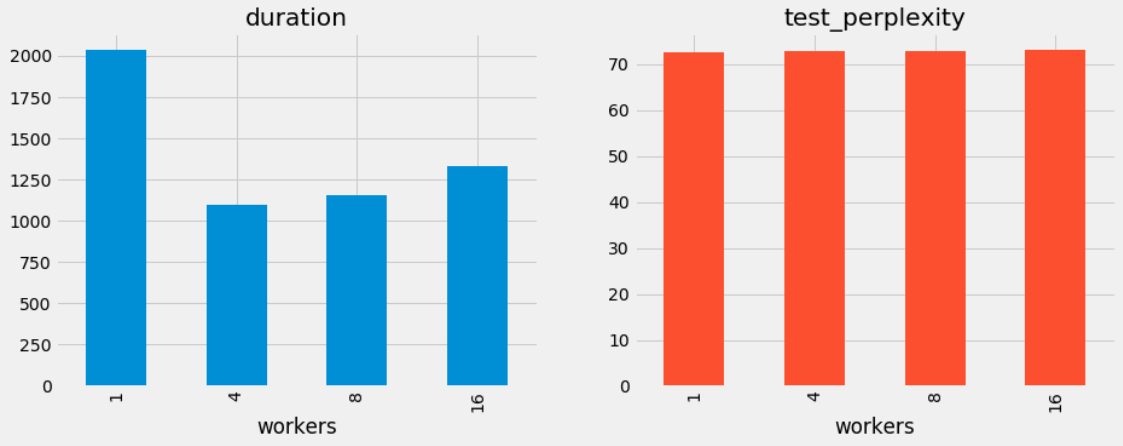
[160]:
```
df = pd.read_excel('timings/timings.xlsx')
df.head()
```

[160]:

	workers	num_topics	duration	test_perplexity
0	1	10	2,035.21	72.62
1	1	50	1,611.58	84.46
2	4	10	1,093.39	72.74
3	4	50	1,067.98	82.12
4	8	10	1,154.87	72.79

[167]:
```
df[df.num_topics==10].set_index('workers')[['duration', 'test_perplexity']].
 ↪plot.bar(subplots=True, layout=(1,2), figsize=(14,5), legend=False)
```

[167]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f7bf634ae48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f7f14ad5080>]],
      dtype=object)
```


duration
2000
1750
1500
1250
1000
750
500
250
0
1
4
8
16
workers
test_perplexity
70
60
50
40
30
20
10
0
1
4
8
16
workers