task01-initial-topic-investigation

September 16, 2018

loganjtravis@gmail.com (Logan Travis)

```
In [1]: %%capture --no-stdout

        # Imports; captures errors to supress warnings about changing
        # import syntax
        from collections import OrderedDict
        import os, pickle, random
        import gensim.models as models, gensim.matutils as matutils, \
                gensim.corpora as corpora
        import matplotlib.pyplot as plot
        import nltk
        import numpy as np
        import pandas as pd
        import pyLDAvis.gensim
        from scipy.sparse import load_npz, save_npz
        from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [2]: # Set random seed for repeatability
        random.seed(42)
```

```
In [3]: # Set matplotlib to inline to preserve images in PDF
        %matplotlib inline
```

1 Summary

From course page Week 1 > Task 1 Information > Task 1 Overview:

> The goal of this task is to explore the Yelp data set to get a sense about what the data look like and their characteristics. You can think about the goal as being to answer questions such as:
>
> 1. What are the major topics in the reviews? Are they different in the positive and negative reviews? Are they different for different cuisines?
> 2. What does the distribution of the number of reviews over other variables (e.g., cuisine, location) look like?
> 3. What does the distribution of ratings look like?
>
> In general, you can address such questions by showing visualization of statistics computed based on the data set or topics extracted from review text.

2 Grading Rubric

From course page Week 1 > Task 1 Information > Task 1 Rubric:

> You will evaluate your peers' submission for Task 1 using this rubric. While evaluating, consider the following questions:
>
> - Application of a topic model: Was the description of the topic modeling procedure clear enough such that you can produce the same results?
> - Topic visualization: Does the topic visualization effectively display the data?
> - Data exploration: Was the description of the two sets of data they selected for comparison clear enough to follow?
> - Visualization comparison: Does the visualization component highlight the differences/similarities between the data?
>
> Note that the examples listed in the "Excellent" column are not an exclusive list for each category. You may choose to award 6 points for any effort in your peers' submissions that goes beyond what is required.

Criteria	Poor (1 point)	Fair (3 points)	Good (5 points)	Excellent (6 points)
Task 1.1: Application of a topic model	A topic model was either not used or did not generate any topic.	A topic model was used, but the report fails to mention what model was used and/or how it is applied to the data set.	The report clearly explains what topic model was used and how it was applied to the data set.	For example, multiple topic models were used and the report analyzes the differences between them.
Task 1.1: Generated visualization	The visualization is either absent or useless.	The visualization is present but does not help make clear what topics the people have talked about in the reviews.	The visualization clearly shows and distinguishes what topics people have talked about in the reviews.	For example, multiple visualizations were used and the report analyzes the comparative strengths of each.

Criteria	Poor (1 point)	Fair (3 points)	Good (5 points)	Excellent (6 points)
Task 1.2: Generated sets of topics	The two subsets are not comparable.	The two subsets are comparable. A topic model was used on the two subsets, but the report fails to mention what model was used and/or how it was applied to the data set.	The two subsets are comparable. The report clearly explains what topic model was used and how it was applied to the two subsets.	For example, multiple interesting subsets were identified and assessed for their usefulness, or multiple topic models were applied to the two subsets with differences between them analyzed.
Task 1.2: Visualization of comparison	The two subsets are visualized in such a way that similarities and differences are not clear.	The two subsets are visualized in such a way to show the similarity of the two subsets, but no attempt was made to show the differences.	The two subsets are visualized in such a way that both similarities and differences are very apparent.	Extra transformation of the data was done to improve visualization, or multiple ways of visualizing the topics were used to provide a very comprehensive comparison.
Visualizations: Appropriate-ness of choice	The visualization methods are not suitable for the type of data.	The visualization methods are suitable for the type of data, but another way to visualize the data is clearly better.	The visualization methods used are quite suitable for the type of data and made relationships clear.	Furthermore, extra effort was made to make the visualizations beautifully designed and/or usefully interactive.

3 Get Yelp Review Data Set

I cleaned the Yelp review data in a separate notebook to shorten this report. "Cleaned" only means:

- Read the JSON file into a Pandas dataframe
- Expanded the `votes` feature from nested JSON into separate features:
  - `votes_cool`

- votes_funny
- votes_useful

- Saved the final data frame to a pickle for easy loading

```
In [4]: # Set paths to data source, work in process ("WIP"), and output
        PATH_SOURCE = "source/"
        PATH_WIP = "wip/"
        PATH_OUTPUT = "output/"

        # Set file paths
        PATH_SOURCE_YELP_REVIEWS = PATH_SOURCE + \
                "yelp_academic_dataset_review.pkl.gzip"
        PATH_WIP_TOKENIZER = PATH_WIP + "task01_tokenizer.pkl"
        PATH_WIP_TOKEN_MATRIX = PATH_WIP + "task01_token_matrix.npz"
        PATH_WIP_LDA_MODEL = PATH_WIP + "task01_lda_model"
        PATH_WIP_LDA_MODEL_VIS = PATH_WIP + "task01_lda_model.html"
```

```
In [5]: # Read pickled dataframe
        dfYelpReviews = pd.read_pickle(PATH_SOURCE_YELP_REVIEWS)
```

```
In [6]: # Print dataframe shape and head
        print(f"Shape: {dfYelpReviews.shape}")
        dfYelpReviews.head()
```

```
Shape: (1125458, 9)
```

```
Out[6]:                                    business_id       date  stars  \
        review_id
        15SdjuK7DmYqUAj6rjGowg  vcNAWiLM4dR7D2nwwJ7nCA 2007-05-17      5
        RF6UnRTtG7tWMcrO2GEoAg  vcNAWiLM4dR7D2nwwJ7nCA 2010-03-22      2
        -TsVN230RCkLYKBeLsuz7A  vcNAWiLM4dR7D2nwwJ7nCA 2012-02-14      4
        dNocEAyUucjT371NNND41Q  vcNAWiLM4dR7D2nwwJ7nCA 2012-03-02      4
        ebcN2aqmNUuYNoyvQErgnA  vcNAWiLM4dR7D2nwwJ7nCA 2012-05-15      4

                                                                             text  \
        review_id
        15SdjuK7DmYqUAj6rjGowg  dr. goldberg offers everything i look for in a...
        RF6UnRTtG7tWMcrO2GEoAg  Unfortunately, the frustration of being Dr. Go...
        -TsVN230RCkLYKBeLsuz7A  Dr. Goldberg has been my doctor for years and ...
        dNocEAyUucjT371NNND41Q  Been going to Dr. Goldberg for over 10 years. ...
        ebcN2aqmNUuYNoyvQErgnA  Got a letter in the mail last week that said D...

                                  type                 user_id  votes_cool  \
        review_id
        15SdjuK7DmYqUAj6rjGowg  review  XqdODzHaiyRqVH3WRG7hzg           1
        RF6UnRTtG7tWMcrO2GEoAg  review  H1kH6QZV7Le4zqTRNxoZow           0
        -TsVN230RCkLYKBeLsuz7A  review  zvJCcrpm2yOZrxKffwGQLA           1
```

```
dNocEAyUucjT371NNND41Q  review  KBLW4wJA_fwoWmMhiHRVOA           0
ebcN2aqmNUuYNoyvQErgnA  review  zvJCcrpm2yOZrxKffwGQLA           1

                        votes_funny  votes_useful
review_id
15SdjuK7DmYqUAj6rjGowg            0             2
RF6UnRTtG7tWMcrO2GEoAg            0             2
-TsVN23ORCkLYKBeLsuz7A            0             1
dNocEAyUucjT371NNND41Q            0             0
ebcN2aqmNUuYNoyvQErgnA            0             2
```

4 Calculate TF-IDF

I separated calculating TF-IDF from building models. The GenSim package includes several models (e.g., LDA, random projections, and LSI) that all accepts a TF-IDF matrix as input. **However**, I had to make early parameter decisions to keep the data size manageable (using an AWS EC2 t3.large instance). My first LDA model ran out of memory!

I therefore tested againts a tiny subset (0.1%) of the Yelp reviews. After producing several LDA models (my focus), I determined a reasonble set of constraints for TF-IDF:

- Limit maximum tokens (more on "tokens" versus "words" below) to 10,000. This is an extreme upper limit. The SciKit Learn `TfidfVectorizer` class (link to documentation) never yielded more than 5,000 tokens based on my other parameters. Most runs identified approx. 1,000 tokens.
- Exclude tokens appearing in more than 50% of documents. These add little value for differntiating topics.
- Exclude tokens appearing in less than 1% of documents. I tested many settings for this parameter ranging down to 2 documents and up to 10% of all documents. The Yelp reviews include numerious limited-use terms (e.g., people and place names) and I found it difficult to interpret the topics with too many present.

```
In [7]: # Set flag to load token matrix from file if found; set this to
        # False when changing other parameters
        load_token_matrix_from_file = False
        overwrite_saved_token_matrix = not load_token_matrix_from_file

        # Set token limit
        max_features = 10000

        # Set document frequency ceiling; topic analysis will ignore
        # words found in more documents
        max_df = 0.5

        # Set document frequency floor; topic analysis will ignore
        # words found in fewer document
        min_df = 0.01
```

4.1 Custom Tokenizer

The SciKit Learn `TfidfVectorizer` class has a default pre-processor and tokenizer. While the pre-processing steps met my needs the tokenizer did not lemmatize nor stem words. Those two additional steps produced more stable topics. I therefore created my own tokenizer.

Note: I created `MyTokenizer` as a class to internalize instantiation of NLK's `WordNetLemmatizer` (link to documentation).

```
In [8]: class MyTokenizer:
            def __init__(self):
                """String tokenizer utilizing lemmatizing and stemming."""
                self.wnl = nltk.stem.WordNetLemmatizer()

            def __call__(self, document):
                """Return tokens from a string."""
                return [self.wnl.lemmatize(token) for \
                                token in nltk.word_tokenize(document)]
```

```
In [9]: # Create TF-IDF vectorizer
        vectorizer = TfidfVectorizer(max_features=max_features, \
                                     max_df=max_df, min_df=min_df, \
                                     stop_words="english", \
                                     use_idf=True, \
                                     tokenizer=MyTokenizer())
```

4.2 Excessive Data Set Size

Setting parameters to limit excessively in/frequent tokens helped to manage overall data size. Unfortunately, both proved insufficient to permit the LDA model to fit within the memory available on my machine. I therefore worked on a 30% sample of the Yelp review data set.

```
In [10]: # Set working dataframe to a 30% sample of the full data set;
         # too large otherwise
         df = dfYelpReviews.sample(frac=0.3)
```

```
In [11]: %%time

         # Load token matrix and vectorizer from file if found and
         # flag set to permit; otherwise calculate new TF-IDF
         tokenMatrix = None
         if(load_token_matrix_from_file and \
            os.path.isfile(PATH_WIP_TOKEN_MATRIX) and
            os.path.isfile(PATH_WIP_TOKENIZER)):
             print(f"Loading token matrix from file \"{PATH_WIP_TOKEN_MATRIX}\"...")
             tokenMatrix = load_npz(PATH_WIP_TOKEN_MATRIX)

             print(f"Loading vectorizer from file \"{PATH_WIP_TOKENIZER}\"...")
             f = open(PATH_WIP_TOKENIZER, "rb")
             vectorizer = pickle.load(f)
            f.close()
        else:
            print("Calculating TF-IDF to build token matrix...")
            tokenMatrix = vectorizer.fit_transform(df.text)
            overwrite_saved_token_matrix = True
```

```
Calculating TF-IDF to build token matrix...
CPU times: user 7min 49s, sys: 6.33 s, total: 7min 55s
Wall time: 8min 8s
```

```
In [12]: # Print token matrix shape
         print("Found {0[1]:,} tokens in {0[0]:,} documents".format(tokenMatrix.shape))
```

```
Found 896 tokens in 337,637 documents
```

```
In [13]: # Save token matrix and vectorizer to file if changed
         if(overwrite_saved_token_matrix):
             save_npz(PATH_WIP_TOKEN_MATRIX, tokenMatrix)
             f = open(PATH_WIP_TOKENIZER, "wb")
             pickle.dump(vectorizer, f)
             f.close()
```

5 Find Topics Using LDA

I focused on Latent Dirichlet Allocation. I used it in other Coursera classes and feel comfortable with its basic premise. Similar to calculating TF-IDF, I adjusted several parameters using a tiny subset (0.1%) of the Yelp reviews:

- Set number of topics to 50
- Display topics using their 5 most frequent tokens

```
In [14]: # Set flag to load LDA model from file if found; set this to
         # False when changing other parameters
         load_lda_model_from_file = True and load_token_matrix_from_file
         overwrite_saved_lda_model = not load_lda_model_from_file

         # Set number of topics
         num_topics = 50

         # Set number of words to display for each topic
         num_words = 5
```

```
In [15]: # Convert GenSim corpus from token matrix
         corpus = matutils.Sparse2Corpus(tokenMatrix, documents_columns=False)
```

```
In [16]: # Create a GenSim dictionary for documents; Note: Passes the
         # vectorizer tokens as a single "document".
         dictionary = corpora.Dictionary([vectorizer.get_feature_names()])
```

```
In [18]: %%time
         %%capture --no-stdout

         # Load LDA model form file if found and flag set to permit;
         # otherwise find topics
         lda = None
         if(load_lda_model_from_file and \
            os.path.isfile(PATH_WIP_LDA_MODEL)):
             print(f"Loading LDA model from \"{PATH_WIP_LDA_MODEL}\"...")
             lda = models.ldamulticore.LdaMulticore.load(PATH_WIP_LDA_MODEL)
         else:
             print("Finding topics using LDA...")
             lda = models.ldamulticore.LdaMulticore(corpus, \
                                                    num_topics=num_topics,
                                                    id2word=dict(dictionary.items()))
             overwrite_saved_lda_model = True

Finding topics using LDA...
CPU times: user 1min 20s, sys: 11.3 s, total: 1min 31s
Wall time: 1min 32s
```

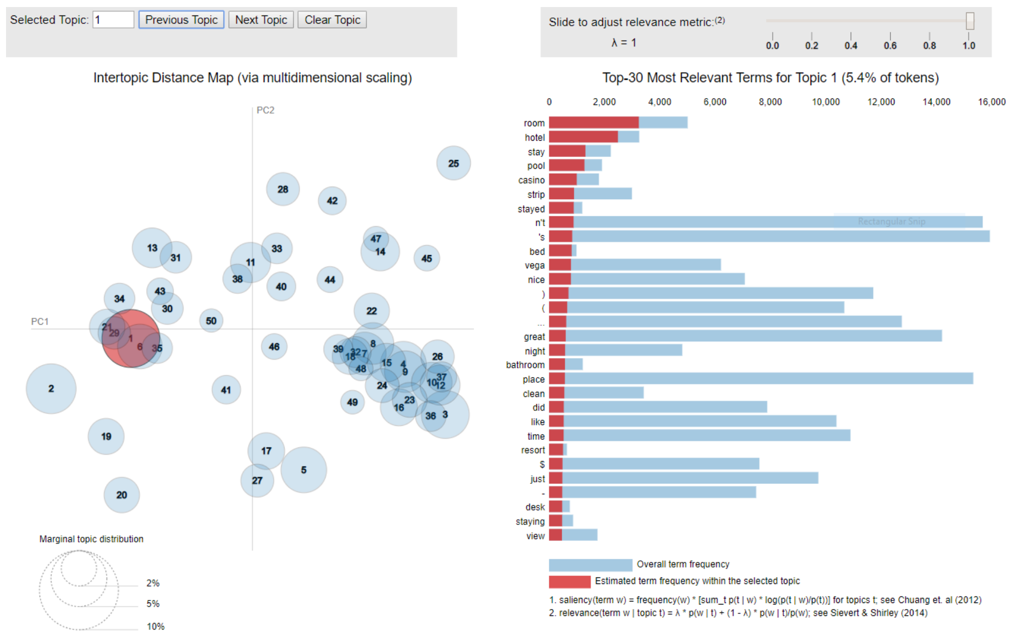
```
In [19]: # Print a few topics for sanity check
         lda.show_topics(num_topics=num_topics, num_words=num_words)[:5]

Out[19]: [(0,
           '0.065*"coffee" + 0.018*"tea" + 0.012*"place" + 0.012*"\'s" + 0.012*"shop"'),
          (1,
           '0.015*"helped" + 0.011*"foot" + 0.011*"great" + 0.010*"time" + 0.009*"funny"'),
          (2,
           '0.031*"room" + 0.026*"hotel" + 0.018*"casino" + 0.014*"strip" + 0.013*"stay"'),
          (3,
           '0.088*"pizza" + 0.013*"slice" + 0.012*"crust" + 0.012*"\'s" + 0.012*"pie"'),
          (4,
           '0.027*"mexican" + 0.025*"salsa" + 0.023*"chip" + 0.021*"margarita" + 0.017*"food"')]
```

```
In [20]: # Save LDA model to file if changed
         if(overwrite_saved_lda_model):
             lda.save(PATH_WIP_LDA_MODEL)
```

6 Graphing Topics from LDA

Priting the most frequent tokens of topics as I did above provides a useful sanity check. However, it does not clarify the relation between topics nor provide a means of exploring them. The *awesome* pyLDAvis package does and includes easy integration with GenSim models.

…but I cannot save the output to PDF! I encourage you to check ouT pyLDAvis. For this assignment, I embedded representative images.


Selected Topic: 1
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0 0.2 0.4 0.6 0.8 1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
Top-30 Most Relevant Terms for Topic 1 (5.4% of tokens)
0 2,000 4,000 6,000 8,000 10,000 12,000 14,000 16,000
room
hotel
stay
pool
casino
strip
stayed
n't
's
bed
vega
nice
)
(
...
great
night
bathroom
place
clean
did
like
time
resort
$
just
-
desk
staying
view
Marginal topic distribution
2%
5%
10%
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

lda-model-example-hotel

```
In [21]: %%capture --no-stdout

         # Prepare visualization
         vis = pyLDAvis.gensim.prepare(lda, corpus, dictionary)
```

```
In [22]: # Save visualization to file for later image extraction then
         # display inline; inline display will not save to PDF
         pyLDAvis.save_html(vis, PATH_WIP_LDA_MODEL_VIS)
         # pyLDAvis.display(vis)
```

6.1 Most Topics Suggest a Type of Business

The top terms for most topics suggest a type of business such as hotel or mexican restaurant. Each topic includes keywords for both positive and negative sentiment. Please see examples below. Such topics intuitively makes sense to me. Most Yelp reviews describe the business and its services. Those terms often overwhelm sentiment-related terms.

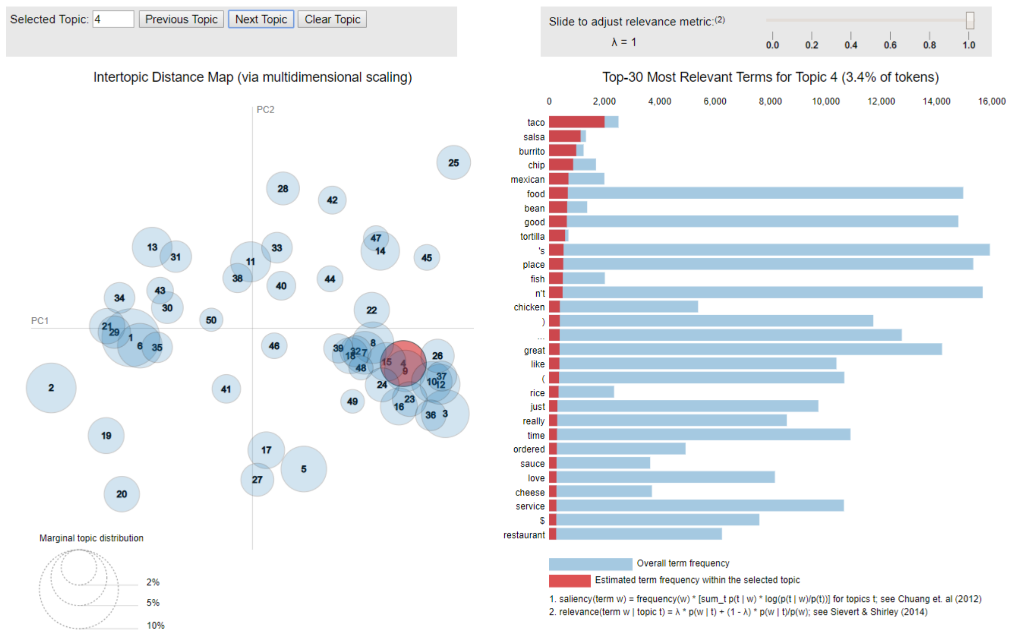
Note: I apologize for images appearing on later pages. I spent 30 minutes trying to get the Jupyter save to PDF working. This was my best result.

6.1.1 Example Topic Suggesting Hotel

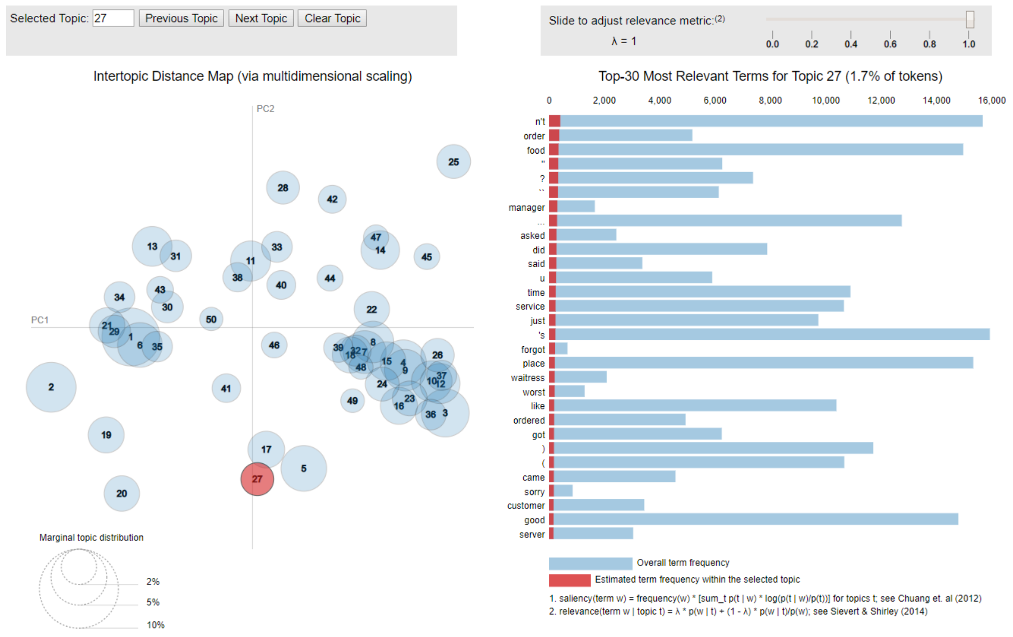
6.1.2 Example Topic Suggesting Mexican Cuisine

6.2 Not All Topics Make Obvious Sense

The top terms for a topic do not always suggest a real-world categorization. The example below includes one or two terms related to but many more relating to sentiment. I suspect such topics capture terse reviews such as "great food" and "terrible service."


Selected Topic: 4
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0 0.2 0.4 0.6 0.8 1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 4 (3.4% of tokens)
0 2,000 4,000 6,000 8,000 10,000 12,000 14,000 16,000
taco
salsa
burrito
chip
mexican
food
bean
good
tortilla
's
place
fish
n't
chicken
)
...
great
like
(
rice
just
really
time
ordered
sauce
love
cheese
service
$
restaurant
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

lda-model-example-mexican-cuisine


Selected Topic: 27
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0 0.2 0.4 0.6 0.8 1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
Marginal topic distribution
2%
5%
10%
Top-30 Most Relevant Terms for Topic 27 (1.7% of tokens)
0 2,000 4,000 6,000 8,000 10,000 12,000 14,000 16,000
n't
order
food
"
?
"
manager
...
asked
did
said
u
time
service
just
's
forgot
place
waitress
worst
like
ordered
got
)
(
came
sorry
customer
good
server
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

lda-model-example-unclear

7 Compare Useful Versus Not Useful Reviews

Each review in the Yelp data set includes votes for cool, funny, and useful. I thought comparing topics generated from useful versus not useful would prove... useful!

```
In [23]: # Get indices for Yelp reviws split by useful versus not
         # useful reviews
         df["row_num"] = range(df.shape[0])
         idxUseful = list(df["row_num"][df["votes_useful"] > 0])
         idxNotUseful = list(df["row_num"][df["votes_useful"] == 0])
```

```
In [24]: # Create separate corpi from useful versus not useful reviews
         corpusUseful = matutils.Sparse2Corpus(tokenMatrix[idxUseful, :], \
                                               documents_columns=False)
         corpusNotUseful = matutils.Sparse2Corpus(tokenMatrix[idxNotUseful, :], \
                                                  documents_columns=False)
```

```
In [25]: # Print the number of reviews in each corpus
         print("The data set has {:,} useful and {:,} not useful reviews.".format(len(corpusUsef
```

```
The data set has 165,231 useful and 172,406 not useful reviews.
```

The subsets of useful and not useful reviews appear roughly equal. Their similar size should improve the comparability of the resulting topic models.

7.1 Find Topics Using LDA

I stuck with LDA for finding topics in useful and not useful subsets. It makes results easier to compare to the topics generated from the full dat set.

```
In [26]: %%time
         %%capture --no-stdout

         # Find topics from useful reviews
         print("Finding topics from useful reviews...")
         ldaUseful = models.ldamulticore.LdaMulticore(corpusUseful, \
                                                      num_topics=num_topics, \
                                                      id2word=dict(dictionary.items()))
```

```
Finding topics from useful reviews...
CPU times: user 48.6 s, sys: 11.9 s, total: 1min
Wall time: 59.1 s
```

```
In [27]: %%time
         %%capture --no-stdout

         # Find topics from not useful reviews
```

```
        print("Finding topics from not useful reviews...")
        ldaNotUseful = models.ldamulticore.LdaMulticore(corpusNotUseful, \
                                                        num_topics=num_topics,
                                                        id2word=dict(dictionary.items()))
```

```
Finding topics from not useful reviews...
CPU times: user 30.6 s, sys: 9.11 s, total: 39.8 s
Wall time: 39.1 s
```

7.2 Graph Terms, Topics, and Models

The pyLDAvis package does not help compare different topic models. I instead graphed the top terms across topics and models. Side-by-side comparison of the term distributions revealed similarities and differences between models.

```
In [30]: # Complile a set of all top-N tokens from each topic
         token_set = set()
         getTopNTokens = lambda m, topn: [w[0] for t in range(m.num_topics) \
                                         for w in m.get_topic_terms(t, topn)]
         token_set.update(getTopNTokens(ldaUseful, num_words))
         token_set.update(getTopNTokens(ldaNotUseful, num_words))
```

```
In [31]: # Create a dataframe from token IDs to more easily
         # manage related data
         dfCompare = pd.DataFrame(list(token_set), columns=["token_id"])
```

```
In [32]: # Add column for token from ID
         dfCompare["token"] = dfCompare.token_id.apply(lambda i: dictionary[i])
```

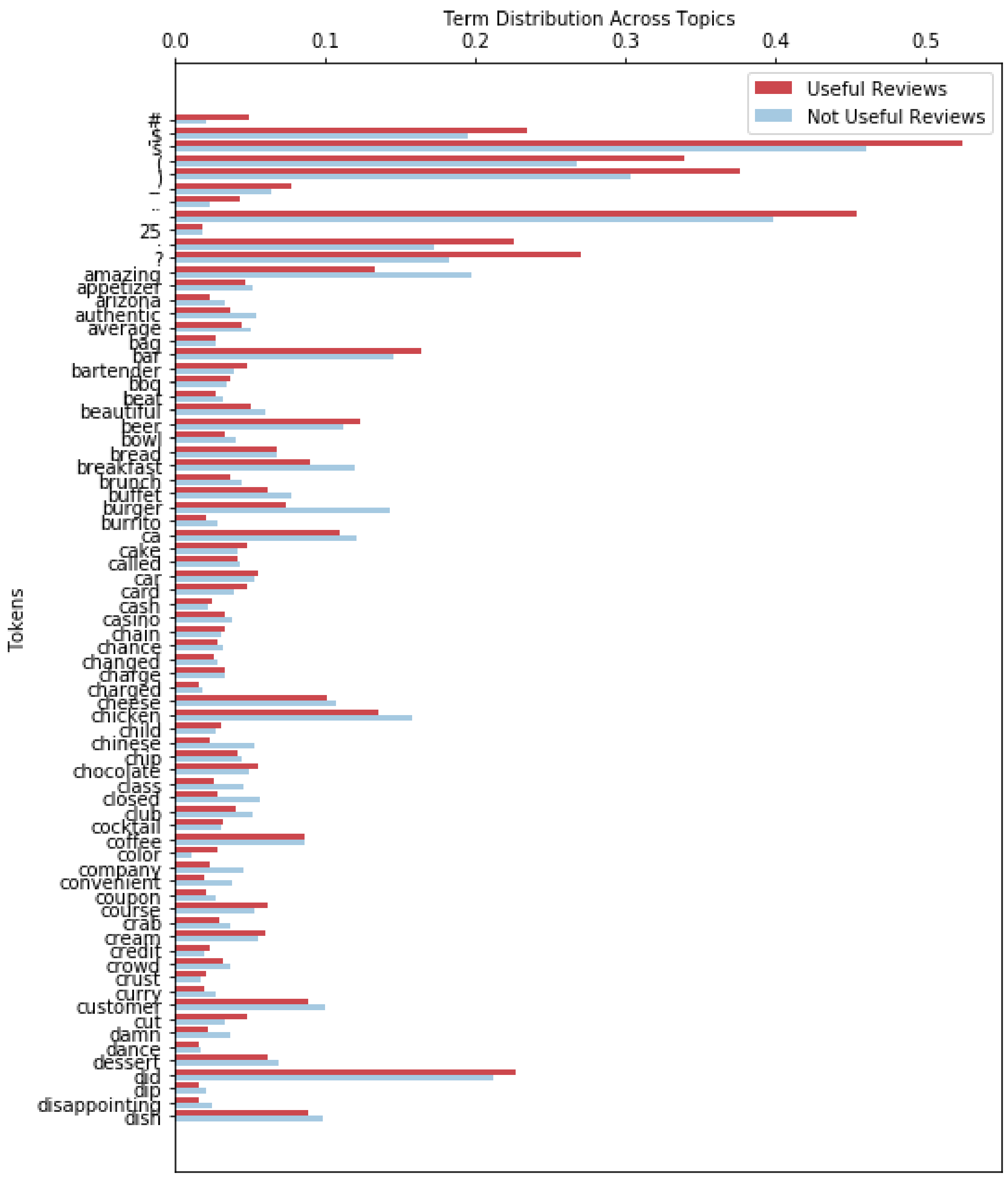
```
In [33]: # Create functions to count number of topics and sum
         # distribution by token
         #
         # Note: Set `minimum_probability` parameter for `get_term_topics`
         # to zero because default must be much higher. Not documented.
         countTermTopics = lambda w, m: len(m.get_term_topics(w, 0))
         sumTermTopicDist = lambda w, m: 0 if m.get_term_topics(w, 0) is None else \
                                        sum([t[1] for t in m.get_term_topics(w, 0)])
```

```
In [34]: # Add columns counting and summing distributions for
         # useful topics
         dfCompare["count_useful"] = dfCompare.token_id.apply(lambda i: countTermTopics(i, ldaUs
         dfCompare["dist_useful"] = dfCompare.token_id.apply(lambda i: sumTermTopicDist(i, ldaUs
```

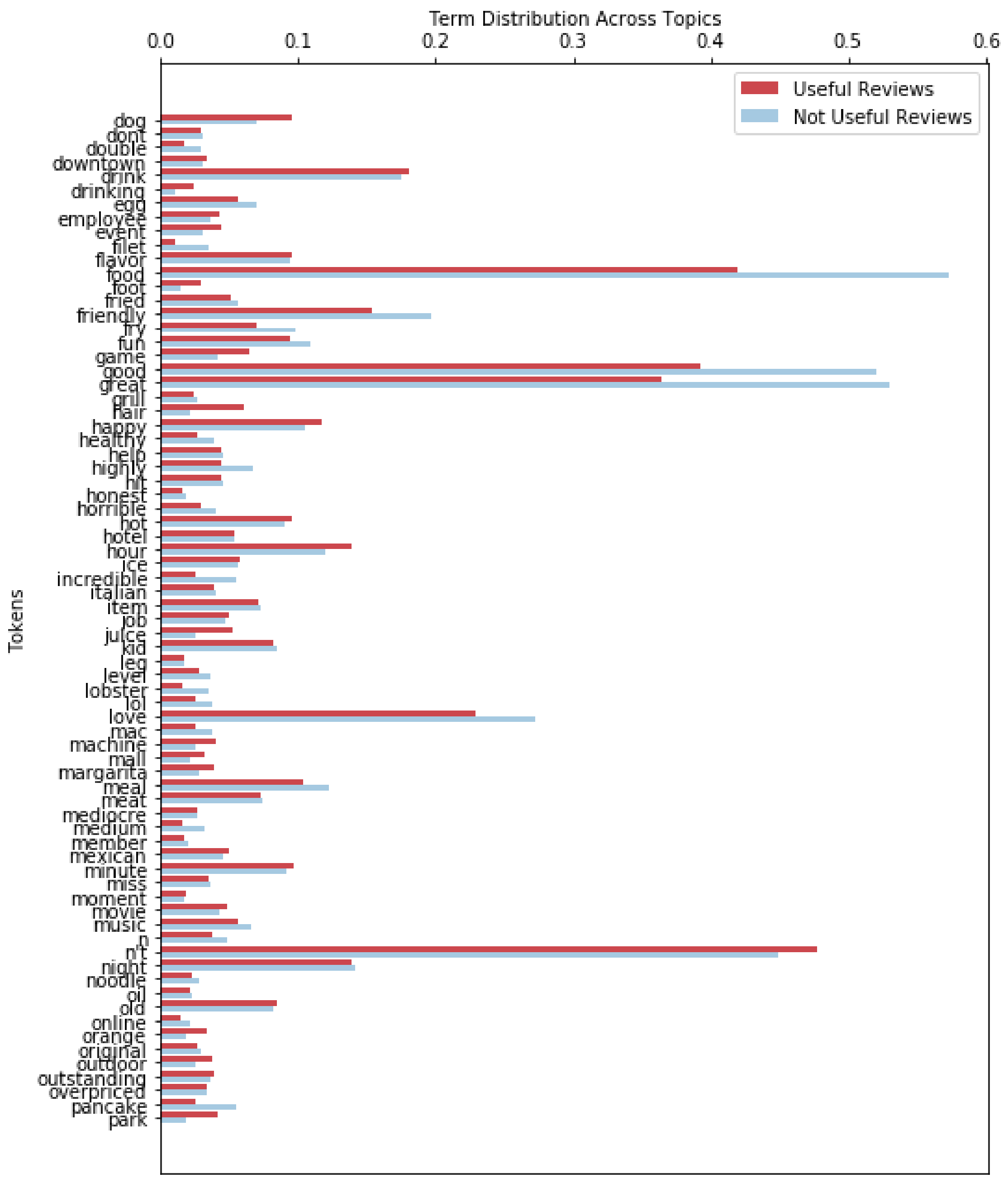
```
In [35]: # Add columns counting and summing distributions for
         # not useful topics
         dfCompare["count_not_useful"] = dfCompare.token_id.apply(lambda i: countTermTopics(i, l
         dfCompare["dist_not_useful"] = dfCompare.token_id.apply(lambda i: sumTermTopicDist(i, l
In [36]: # Set tokens and reset index
         dfCompare.sort_values("token", ascending=False, inplace=True)
         dfCompare.reset_index(drop=True, inplace=True)
```

```
In [41]: # A single chart, while easier to view in Jupyter, looks terrible
         # as a PDF. The helper function below will create the same chart
         # for a subset of tokens.
         def splitGraph(startTokenID, endTokenID):
             """Create chart for subset of useful versus not useful term-topic distribution"""
             fig, ax = plot.subplots()
             fig.set_size_inches(8, 11)

             # Add horizontal bars
             index = np.array(dfCompare.index)[startTokenID:endTokenID]
             width = 0.5
             gap = width / 5
             barsUseful = ax.barh(index+width, \
                                  list(dfCompare.dist_useful)[startTokenID:endTokenID], \
                                  width-gap, label="Useful Reviews", align="edge", \
                                  color="#cc474d")
             barsNotUseful = ax.barh(index+gap,
                                     list(dfCompare.dist_not_useful)[startTokenID:endTokenID], \
                                     width-gap, label="Not Useful Reviews", align="edge", \
                                     color="#a5c9e1")

             # Configure horizontal axis at top
             ax.set_xlabel("Term Distribution Across Topics")
             ax.xaxis.set_label_position("top")
             ax.xaxis.tick_top()

             # Configure vertical axis with token labels
             ax.set_ylabel("Tokens")
         #     ax.set_ylim(2*width-1, len(index))
             plot.yticks(index+width, dfCompare.token[startTokenID:endTokenID])

             # Enable legend
             ax.legend(loc=1)

             # Return plot
             return plot
```

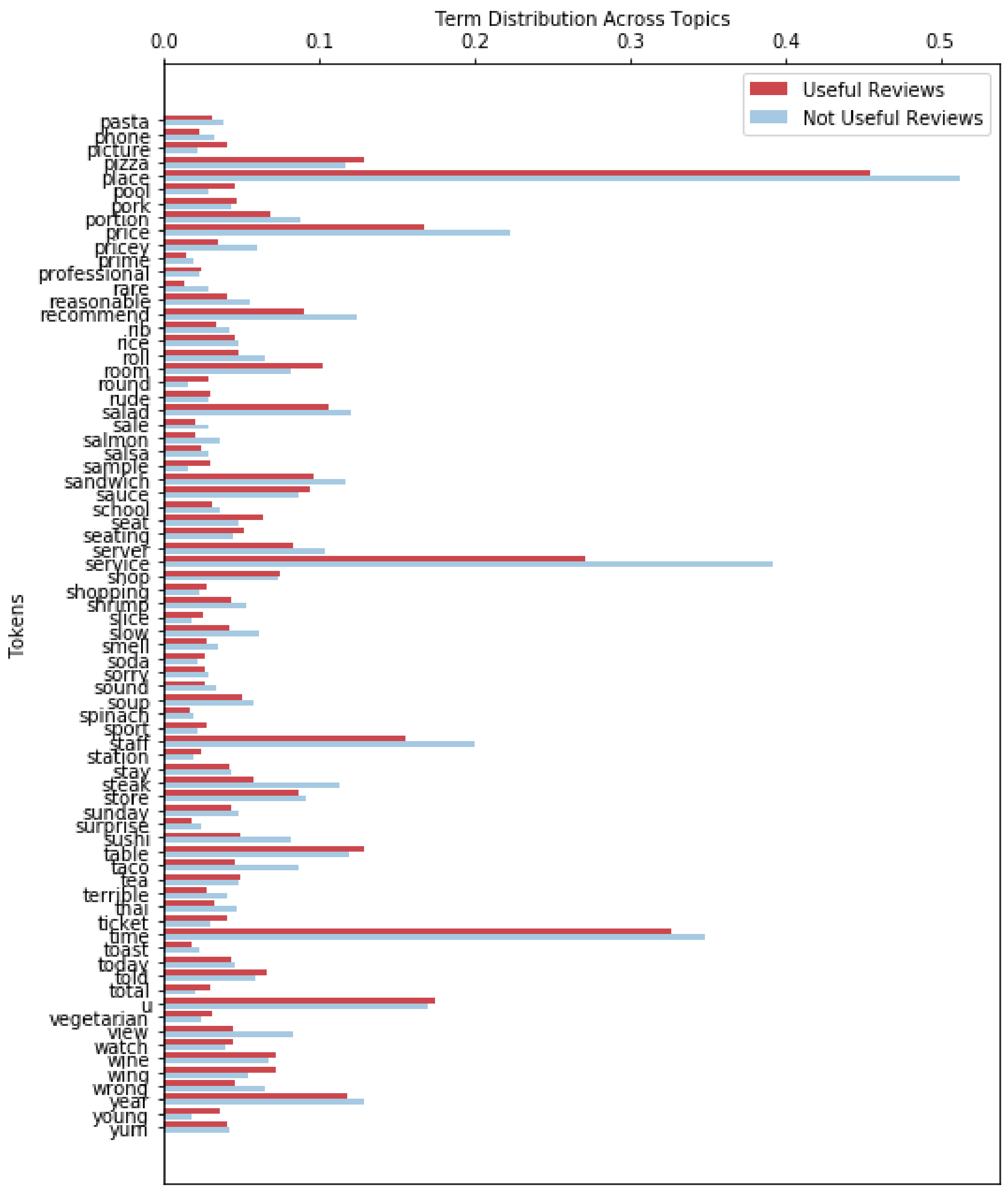
```
In [42]: # First subset
         step = int(dfCompare.shape[0] / 3)
         stop = dfCompare.shape[0]
         start = stop - step
         plot.show(splitGraph(start, stop))
```



```
In [43]: # Second subset
         stop = start
         start = stop - step
         plot.show(splitGraph(start, stop))
```



```
In [44]: # Third (final) subset
         stop = start
         start = 0
         plot.show(splitGraph(start, stop))
```



The resulting models are similar but not identical. They differ most significantly on semantic words like “good” and “love”. They also differ on low-utility words like “place”. Interestingly, the not useful reviews include both low-utility *and semantic* words as a higher proportion of their topic distributions.

The higher proportion of low-utililty words makes sense; generic reviews should garner few votes for usefulness. I do not fully understand the higher proportion of semantic words. Perhaps the difference relates to my earlier supposition on “terse” reviews. Such reviews (e.g., “great food” and “terrible service”) might receive fewer votes as useful.