Name 1: Yuxiang Wang **Name 2: Yuanyuan Qu** **Name 3: Chao Wu**
Student ID 1: 476656 **Student ID : 476823** **Student ID 3:474558**

DAT 562 Final Project Answer Sheet

1. Data Description:

The dataset we use is *Yelp Reviews*. It is collected by Yelp engineers at 2017 and it contains more than 115 million reviews in total. There are five json files in the dataset, including the files of business, user, check-in, tip and review. In our project, we mainly focus on the review file.

In the review file, there are 10 columns in total, which include review id, user id, business id, stars, date, useful, funny, cool, text and type. The three id columns contain unique id for each review, user and business. The stars column refers to the stars rating that customers labeled at Yelp, which ranges from 1 to 5. The date column refers to the date that customers wrote their review, which ranges from 2004 to 2017. The funny, cool and useful columns are customers' label towards other customers'' reviews or tips. The text column is the review that we are going to mainly focus on. The type column in the review file has only one value, which is review.

2. Project Objectives:

This project is aiming at two main objectives achieved by topic modeling and sentiment analysis.

Trend analysis:

We would like to investigate the customers' preference change in Yelp reviews over time by analyzing groups of top keywords in topics using LDA topic modeling. We divide the sample dataset into 3 time ranges from 2004 to 2017 and using unsupervised training to assign topics to groups of words. We plan to distinguish most popular topics in certain period and make recommendations both for Yelp and merchants to help them better understand and attract customers. Here is a potential outcome, 5 years ago, people write reviews revealing food and service of the restaurant, 5 years later, they may pay more attention to the decoration and environment.

Sentiment Analyzer Training:

We applied VADER Lexicon-Based Sentiment Analysis as unsupervised learning approach and Naive Bayes Classifier-Based Sentiment Analysis as supervised learning approach to train the analyzers and select the one with highest accuracy rate.

Our objective is to use the most powerful model to predict other unlabeled reviews on other platforms for merchants. For example, some restaurants may have their own accounts on Facebook, they can receive many comments and reviews on that platform but have no idea what the attitude in reviews is without label. In this case, our model will be extremely useful in helping them interpret, understand and even build a closer relationship with customers. For instance, They could use this model to monitor their restaurants performance very often, since the model greatly reduces the difficulty of reviewing a large amount of text. They could also accurately target the negative reviews towards their restaurants precisely and then respond to the reviews to solve the problem. All in all, we believe our model can be a bridge between customers and merchants.

3. Methodology:

a. Topic Modeling:

Topic modeling is used to detecting word and phrase patterns within the document, and automatically clustering word groups as topics. We use Latent Dirichlet Allocation for topic

modeling. Latent Dirichlet Allocation tries to infer the hidden topic structure from the documents by computing the posterior distribution given by the documents.

In the project, we first set seed and randomly pick 20000 reviews from the review file. Then we divide the dataset according to time range. In particular, Corpus 1 is from 2004 to 2008, corpus 2 is from 2009 to 2013 and corpus 3 is from 2014 to 2017. Next, we set $\lambda = 0.25$ to select 4 topics on each of these three corpuses and identify dominant topics overtime using word weights.

To evaluate the performance of each LDA model, we calculated coherence score both for the whole model and by topic using semantic similarity of words in a topic. At the same time, we refer log-likelihood score to maximize the probability of observed text and perplexity ("confusion") metric to observe the probability to observe a sample text, which is normalized by the number of words in that text.

We use the results to make recommendations for merchants to focus on the most popular topics. Meanwhile, we suggest that Yelp updates its recommendation and involve more trending topics to its search results.

Pros:
Using LDA for topic modeling is interpretable because it calculates probabilities distribution over topics and over words in each topic. We can simply compare weights of words and sort the weights value to get the ranking of top words in each cluster.

Cons:
It is hard to identify which parameters are the best because topics are soft-clustered by unsupervised learning. Although perplexity and coherence score contribute to the feasibility of the model, it is still hard to tell the quality of the model.

b. VADER Lexicon-Based Sentiment Analysis

Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. It's also known as opinion mining, deriving the opinion or attitude of a speaker.

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis. VADER uses a combination of a sentiment lexicon and a list of lexical features (e.g., words) which are generally labeled according to their semantic orientation as either positive or negative. VADER not only tells about the positivity and negativity score but also tells us about how positive or negative a sentiment is.

For yelp dataset, we labeled the yelp reviews' polarity into positive or negative based on stars, converting five-scale format into polarity labels. We assign 1-3 stars as 'positive' and 4-5 as 'negative'.

We randomly select 20,000 reviews from the original dataset as the sample. And split the sample into 16,000 train_reivews and 4,000 test_reviews. We imported the sentiment analysis function SentimentIntensityAnalyzer() to get the sentiment scores of each text according to the VADER lexicon. This function reports the share of review text to "negative", "neutral" and "positive". We define analyze_sentiment_vader_lexicon function to return VADER sentiment score and binary sentiment polarity. The score is normalized to be between -1 to 1.

We set 0.1 as threshold at first, which means the value below 0.1 stands for “negative” polarity and values above 0.1 correspond to “positive polarity”.

We define a function try_threshold_for_accuracy to check the accuracy rate under different threshold and plot the changes in accuracy rate for a range of thresholds from -1 to 1. From the plot we get the best threshold that maximize the accuracy rate. Model under this threshold would be a better fit for restaurant reviews.

Pros:
a). VADER is open source and easily to acquire and replicate.
b). It is among the most comprehensive tools for social media analysis and can be used on unlabeled text. So it would be a good way for us to optimize model and put it into use of reviews on other social media.
c). The model can be trained further with labels. The precision of prediction on polarity would be improved.

Cons:
a). A lexicon may lack of domain specificity.
b). Measuring accuracy is problematic if we do not have labeled test data. But for this project, we used stars as labels. Therefore, we can improve accuracy of the model based on labels.
c). It does not recognize the context and lose some information in texts.
d). It requires additional tools for visualizing output, like pandas etc.

c. Naive Bayes Classifier-Based Sentiment Analysis

Naive-bayes classifier is a powerful and flexible supervised learning technique on text. In this project, we build a multi-class model to predict customers’ attitudes towards merchants by applying this technique.

To be more specific, we first split all the reviews on Yelp into 5 category based on the review’s stars rating and then we randomly select 1000 reviews from each category. Stars rating at Yelp range from one to five, one means most negative and five means the most positive. The reason we would like to get the equal number of reviews in different star category is that we found there are a lot more high-star ratings than low-star ratings. In other words, customers have a higher intention to leave positive reviews on Yelp. So if we just randomly select reviews based on the original dataset, it can cause the problem that we have a huge amount of five-star ratings but only limited one-star ratings. This could greatly decrease our prediction power on the negative review due to lack of training.

Next, we treat stars rating as the label of the text and train our model based on 80% of the random sample we just selected. In other words, we use 800 reviews in each category and have 4000 samples in the training dataset in total. The rest 20% of the sample reviews will be used as test data. By using star rating as the label, we have a much more granular understanding of customers' attitude than only using positive and negative.

Finally, we use MultinomialNB module to predict customers' attitude on test data. For each review, the model will return 5 posterior probabilities for each class. The class corresponding to the largest probability is selected as the classification result.

In a nutshell, by selecting training samples equally in different categories and using 5 star ratings as the label of the text, we built a powerful multi-class sentiment analyzer.

Pros:
Naive Bayes algorithms are one of the most solid algorithms in machine learning field. The algorithms have been proved that can perform well in multi-class analysis. Moreover, Naive Bayes algorithms were based on a simple assumption that features are independent with each other, which make it easy and fast to deal with large data sets with a huge number of features.

The other advantage of this method is that supervised learning can be more flexible than unsupervised learning, which means the model is subjected to our objective. For example, we could use reviews for only restaurant category to train a sentiment classifier, which could be applied to predict restaurants performance only. This model trained by reviews of specific category will have a higher accuracy compared with unsupervised learning.

Cons:
Compared with unsupervised learning, supervised learning techniques are more time consuming and computationally expensive. For example, training a model with 40 thousands reviews may require your computer to allocate more than 16 GB ram. However, most of the Python version could only be allocated up to 8 GB ram. This means we have to find a balance between efficiency and accuracy.

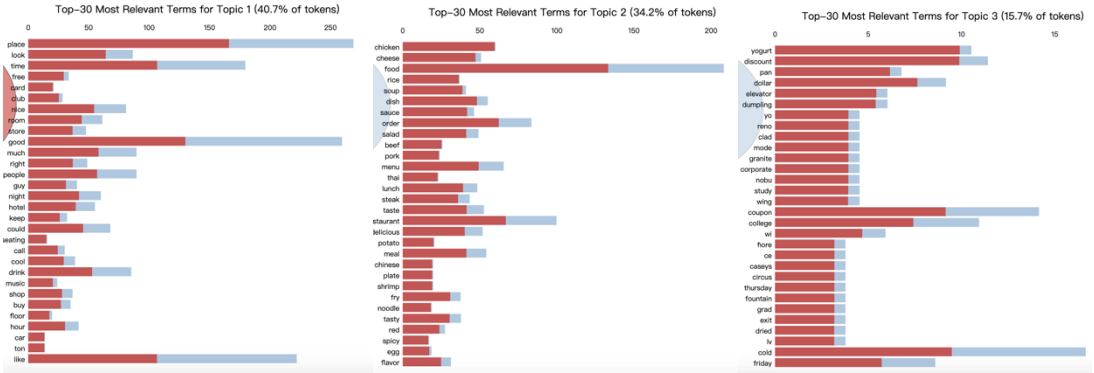
4. Results and Discussion:

a. Topic Modeling:

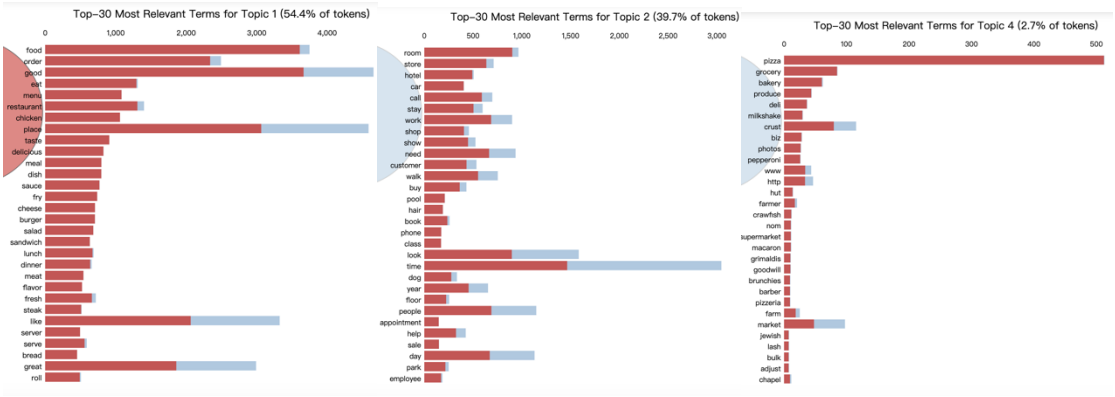
Results:

We choose the relevance metric $\lambda = 0.25$. As there is always one of the topic is German words, we decide to exclude this topic in each corpus modeling.

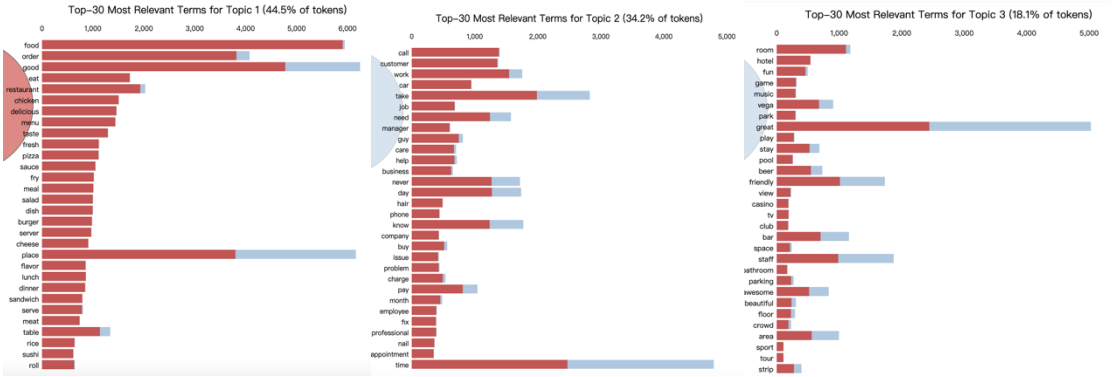
For corpus 1, which ranges from 2004 to 2008, we conclude “hotel” for topic 1, “food” for topic 2. As for topic 3, there is a mix of words from food, money, time and other categories. Words from topic 4 is German and is excluded from analysis.



For corpus 2, which ranges from 2009 to 2013, we conclude “food” for topic 1, “hotel” for topic 2. As for topic 4, “pizza” is way much higher for term frequency with topic and overall term frequency than other words, so we can conclude “deli” for this topic. Topic 3 here is German and is excluded from analysis.



For corpus 3, which ranges from 2014 to 2017, we conclude “food” for topic 1, “hotel” for topic 3. For topic 2, there is a mix of words from categories such as “appointment”, “time” and “service”. Topic 4 here is German and is excluded from analysis.



<u>Topic Modeling for Evaluation:</u>

The results of coherence score(by topic and for the whole model), log-Likelihood and perplexity for each corpus are listed below:

Corpus 1:

Coherence score for the model: -8.4618

Coherence score by topic (higher values are better): [-1.3719 -0.7877 -16.9908 -14.6968]

Log-Likelihood (higher values are better): -242021.92116508883

Perplexity (lower values are better): 2921.594001707484

Corpus 2:

Coherence score for the model: -1.8138

Coherence score by topic (higher values are better): [-1.4651 -3.3716 -1.4989 -0.9196]

Log-Likelihood (higher values are better): -3295949.632462239

Perplexity (lower values are better): 2827.9134912767513

Corpus 3:

Coherence score for the model: -3.4067

Coherence score by topic (higher values are better): [-2.1475 -1.7145 -8.1221 -1.6426]

Log-Likelihood (higher values are better): -4905774.101650478

Perplexity (lower values are better): 2434.479349146027

<u>Discussion:</u>

To avoid violation, we aim to analyze the topics with precise definition of words for each corpus. If considering word weights for each topic, we find the highest coherence scores by topic mostly come from topics that we can precisely identify, such as “food” and “hotel”.

For “food”, the top 5 most relevant terms from 2004 to 2008 are chicken, cheese, food, rice, soup. From 2009 to 2013, the terms are food, order, good, eat, menu. From 2014 to 2017, the terms are food, order, good, eat, restaurant. This trend reveals people are changing their eating behavior from dishes with American style to general quality of food. Furthermore, the rank and coherence score for “fresh”, “fry”, “delicious” is going up while the term “sushi” first shows up in corpus 3, indicating that people are open to variety of fresh and delicious meal regardless of certain kinds of dishes. Restaurants owners and managers of catering industries are recommended to embrace these changes and make reactions.

For “hotel”, the top 5 most relevant terms from 2004 to 2008 are place, look, time, free, card. From 2009 to 2013, the terms are room, store, hotel, car, call. From 2014 to 2017, the terms are room, hotel, fun, game, music. It is clear that people valued money and time when referring a place to stay before. Starting from 2009 to 2010, additional services such as comfort, space, convenience, parking, reservations are of much importance to customers. In recent years, customers care experience when buying service. With most frequent terms in topic 3 for corpus 3, we discover that customers prefer great places with friendly staff, additional services like bars clubs and casinos to have fun, modern facilities and awesome views. Executives of hotels may adjust their strategies to cater market needs.

b. VADER Lexicon-Based Sentiment Analysis

a). When we set the threshold at 0.1 automatically, the accuracy rate is 0.767. The prediction table is shown in appendix Table 1.

b). We calculated the Accuracy Rate of Sentiment Polarity Prediction as a function of Threshold for VADER Scores. From the plot in appendix Plot 2, we can see that as threshold increases from -1 to 0.6096, the accuracy rate increases gradually. But after that, the accuracy rate decreases suddenly.

Therefore we set the threshold at 0.6096 and tried to optimize the model. The accuracy increased to 0.78 at 0.6096 threshold. The prediction table is listed in appendix Table 3.

c. Naive Bayes Classifier-Based Sentiment Analysis

Result:

After we train our model based on 4,000 reviews of 5 classes, we then apply the model to predict 1,000 test reviews and calculate the accuracy rate.

At the beginning, we use the stars rating as the prediction result. However, when we calculate the accuracy rate, it is even less than 50%. We then decided to change 5-classes label into 2-classes label, which is negative and positive. We convert 1-3 stars into negative and 4-5 as positive and calculate the accuracy rate again. The accuracy rate has now improved to 80.4%.

Discussion:

As we listed above, we didn’t get a high accuracy rate at the beginning because we found an interesting fact that in most of the cases, the prediction and true label are very close to each other but not exactly the same. For example, a one-star review may be predicted as a two-stars review and a four-stars review can be wrongly categorized as a five-stars review. We suddenly realized that there is bias in the sample. To be more specific, each customer has his

own rating habit. It can be the case that A is an aggressive person and is more inclined to give one-star ratings even though he doesn't have too much to complain about, while B is a nice guy and is more willing to give a higher-star rating even though he wrote tons of things to complain at the review. This bias made our machine to confuse. That's the reason why it may categorize some reviews to a similar but not the right class.

In spite of that, we still think that it is a big improvement to use 5-scale rating as the label to train the model. However, in order to get a more precise result, we need to train more data to minimize the bias. In our project, due to the small ram of our computer, we were only able to train a small size of dataset, which may result the computer confusion at the first time. We believe that with a stronger computer, we could train a better model based on the 5-scale rating.

5. Conclusion:

a. Recommendations

a). Restaurants should care more about the quality and the variety of food. Instead of only having certain types of foods like American style, they should also pay attention to decoration, service and environment of restaurants.

b). Hotels need to shift their attention from cost-efficient to high quality, which contains convenience, good serving, parking, reservations and comfort. In recent years, hotels may attract customers furthermore if they could provide facilities related with entertainment like clubs or casinos.

c). After trying on several sentiment analysis. We found out that by using Naïve Bayes Classifier-Based Sentiment Analysis to train the model, we would get a higher accuracy. By taking use of this method, restaurants can train their own model to predict reviews on social media. The polarity they get is relatively precise.

b. Possible Shortcomings and ways to overcome them

a). For Naïve Bayes Classifier-Based Sentiment Analysis, since it's highly computationally intensive, we were not able to train a large dataset. So the result at the first time is not very accurate. Next time, if we still want to use the supervised learning techniques, we should get a big-ram computer and train a bigger dataset.

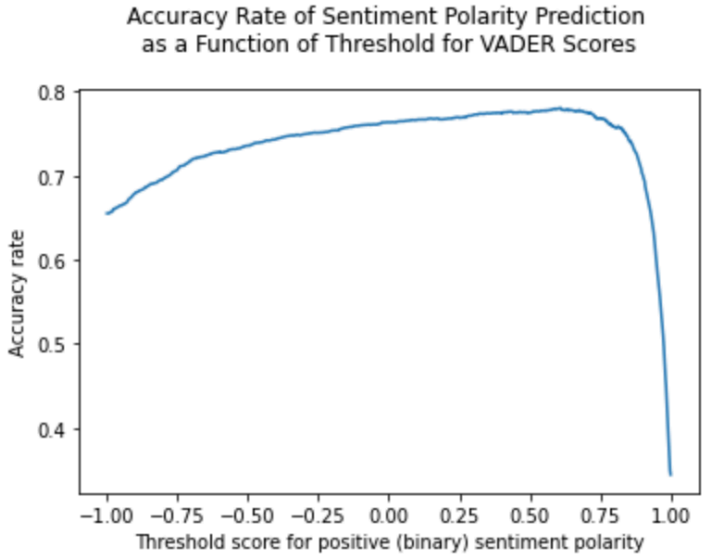
b). Since we would like to apply our trained model based on Yelp to other platform like Facebook, there can be selection bias. The reason is people who write reviews on Yelp may not the same kind of customers on Facebook and their wording could be different. One possible way to deal with this is to add labeled reviews on other platform into train dataset. The model trained in this way could deliver a higher accuracy.

6. Appendices:

a. Tables and Plots

Predicted:	negative	positive	All
True:			
negative	554	825	1379
positive	108	2513	2621
All	662	3338	4000

Table 1 : Crosstable when threshold is 0.1



Plot 2:

Predicted:	negative	positive	All
True:			
negative	735	644	1379
positive	234	2387	2621
All	969	3031	4000

Table 3: CrossTable when threshold is 0.6096

b. Code:

a). Topic Modeling

In [3]:
```
import json
import pandas as pd
import numpy as np
```

In [56]:
```
#OPENING YELP REVIEWS DATASETS
#read the data from disk and split into lines
#we use .strip() to remove the final (empty) line
with open("yelp_academic_dataset_review.json") as f:
    reviews = f.read().strip().split("\n")

#each line of the file is a separate JSON object
reviews = [json.loads(review) for review in reviews]

import random
random.seed(17)
sample = random.sample(reviews, 20000)

```

In [146]:
```
corpus = []
corpus_1 = []
corpus_2 = []
corpus_3 = []
```

In [147]:
```
sample_reviews = pd.DataFrame(sample)
sample_reviews
```

Out[147]:

	review_id	user_id	business_id	stars	date	text	useful	funny	cool	type
0	icFXTyerW1X8p7oSOF0Vog	ojp5d9vW8dQpU12Mv6vthQ	prjaJmqTAnYdeLrWp8_MiA	1	2014-05-22	As bad as a dry cleaning experience could be. ...	4	0	0	review
1	arefzeNBacurVt-gql9J-g	oQ7irCudwDr73-938pXQIg	UV2Jt8slktGu14gLZeNCjA	5	2012-02-18	This restaurant is a gem. The daily brunch is ...	0	0	0	review
2	a-sids8zheK3JBC9DnnR5g	cp0zNS_4chviJdhrSbEdqA	MYn-VzURyyDQqz0l5-PmDQ	5	2016-07-24	I really like the owners!!! I 'used' to like a...	2	1	2	review
3	fVnQtiHIsFHFgaLJT8ULvQ	7hnwrH20r6liJOAlNOPgcw	bMrYaaAy62ppdS09na0RgQ	5	2016-07-07	Love this place. Pizzas are great, the staff i...	0	0	0	review
4	SW65Xgk6A6OpkDE5hCPOHw	ogGW1hddNjqY3LrldLBr4g	vcxvQyAggPqxcHwvJXvjGg	4	2012-04-30	Finally made it out here with a heads up from ...	1	1	1	review

In [148]:
```
for (index, row) in sample_reviews.iterrows():
    date = row['date']
    if '2004' in date or '2005' in date or '2006'in date or '2007'in date or '2008' in date:
        corpus_1.append(row['text'])
    elif '2009' in date or '2010' in date or '2011' in date or '2012' in date or '2013' in date:
        corpus_2.append(row['text'])
    elif '2014' in date or '2015' in date or '2016' in date or '2017' in date:
        corpus_3.append(row['text'])
```

In [149]:
```
corpus_1
```

Out[149]:
```
["This place has a Beer Cave!  Seriously, that's what they call it.  We got to go in and select the beer of our choice from shelves and shelves
es, everything from Bud to Hoegaarden to Chimay and back again.  So many beers I've never even heard of.\n\nAnyway, they have hot dogs too...soo
s.  I came with a group and we all easily satisfied our hot dog cravings.  I had the veggie dog with cheese, tomatoes and dill pickles, and it w
Everyone else was thrilled with their dogs as well.  We also got cheese fries with ranch dipping sauce, fried zucchini, and onion rings...and he
ust kidding, but damn what a greasy meal of awesomeness!",
 'Sat in the 3rd row with the warning that we may get wet.\n\nI got wet.. wet from loving the show that much! (Dirty minds LOL)\n\nI definitely
Cirque du Soleil. I was entertained every minute.',
 "The Tournament of Kings was on of the best things I did in Vegas. My sister suggested we go because she had gone long ago when in High School
t... that alone made me a bit wary of the place. High School... blech! Anyway, despite hitting my head on the table behind me when I first sat d
I'm a big klutz) I ended up having the BEST time. I truly believe this show is what you make of it... so let your inhibitions go and be a big ki
d a rockin' time. My sister, boyfriend and I stuffed our faces with all the food, pounded the tables and yelled at the top of our lungs the enti
ng didn't win but they were all a lot of fun to watch. Psst... Norway and France... get back at me!!! hahahaha...",
 "Terrible coffee.  So bitter I would have to dump a quarter cup of cream and several heaping tablespoons of sugar into this cup-a-joe to overpo
r...  but at that point, why wouldn't I go all out and find myself a milkshake or some other creamy sugary treat?  \r\n\r\nThe muffins are the s
d.  Lots of calorie-laden treats.  I had some carrot cake once and it had that stale refrigerator flavor in the icing.  What a waste. \r\n\r\nMy
cross the street to Starbucks. At least you'll get a mediocre cup of coffee instead of a bad one.",
 "By far the most unique dining expirience I've had in the US!!!!!!!!  I wasn't sure what to make of the menu when my boyfriend made the reserv
soup???  alligator???  .....but I was plesantly suprised at the food, incredible selection and first-rate service.  The decor is classy, and mos
d fairly nicely.  A family or two, but by in large an all adult crowd. Our waiter was informative and fun without being overbearing.  He even br
```

In [153]:
```
#the module 'sys' allows istalling module from inside Jupyter
import sys

!{sys.executable} -m pip install numpy
import numpy as np

!{sys.executable} -m pip install pandas
import pandas as pd

#Natrual Language ToolKit (NLTK)
!{sys.executable} -m pip install nltk
import nltk

!{sys.executable} -m pip install sklearn
from sklearn import metrics
#from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import  CountVectorizer #bag-of-words vectorizer
from sklearn.decomposition import LatentDirichletAllocation #package for LDA

# Plotting tools

from pprint import pprint
!{sys.executable} -m pip install pyLDAvis #visualizing LDA
import pyLDAvis
import pyLDAvis.sklearn

import matplotlib.pyplot as plt
%matplotlib inline

#ignore warnings about future changes in functions as they take too much space
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

#define text normalization function
%run ./Text_Normalization_Function.ipynb #defining text normalization functionaa
```

In [154]:
```
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic %d:" % (topic_idx))
        print(" ".join([feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))

def get_topic_words(vectorizer, lda_model, n_words):
    keywords = np.array(vectorizer.get_feature_names())
    topic_words = []
    for topic_weights in lda_model.components_:
        top_word_locs = (-topic_weights).argsort()[:n_words]
        topic_words.append(keywords.take(top_word_locs).tolist())
    return topic_words
```

For corpus 1

In [155]:
```

normalized_corpus_1 = normalize_corpus(corpus_1)
normalized_corpus_1
```

Out[155]:
```
['place beer cave seriously call select beer choice shelf shelf single bottle everything bud hoegaarden chimay many beer never even heard anyway
many choice group easily satisfy hot dog craving veggie dog cheese tomato dill pickle yummy everyone else thrill dog well cheese fry ranch dip s
ni onion ring heart attack kidding damn greasy meal awesomeness',
 'sat 3rd row warning may wet wet wet love show much dirty mind lol definitely recommend cirque du soleil entertain every minute',
 'tournament king best thing vega sister suggest long ago high school blast alone bit wary place high school blech anyway despite hit head table
it big klutz end best time truly believe show let inhibition big kid rockin time sister boyfriend stuff face food pound table yell top lung ent
n lot fun watch psst norway france hahahaha',
 'terrible coffee bitter dump quarter cup cream several heap tablespoon sugar cup joe overpower flavor point milkshake creamy sugary treat muff
calorie laden treat carrot cake stale refrigerator flavor icing waste advice walk across street starbucks least mediocre cup coffee instead bad
 'far unique din expirience u sure menu boyfriend reservation turtle soup alligator plesantly suprised food incredible selection first rate serv
y people dress fairly nicely family large adult crowd waiter informative fun without overbear even bring us taster turtle soup anxious hesitant
cey rich definatley beware filling date start alligator tender delicious bread bit like calamari entree beautifully present flavorful order crab
nlike place crab cake huge crab crab rest bit veggie compot hold together breading fying way fresh delectable crab meat mistake order entree por
sert specialty bread pudding souflee chicory coffee outstanding top expirience take candy praline way highly recommend update sad true cp tempor
r move aladdin planet hollywood another location strip email however assure open soon boycott vega open',
 'love venetian ive stay many time havent complaint service quality gaming food entertainment cover',
```

In [247]:
```
#define the bag-of-words vectorizer:
bow_vectorizer = CountVectorizer()

#vectorize the normalized data:
bow_corpus_1 = bow_vectorizer.fit_transform(normalized_corpus_1)
```

In [248]:
```
pd.DataFrame(data = bow_corpus_1.todense(), columns = bow_vectorizer.get_feature_names())
```

	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
433	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
434	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
435	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
436	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
437	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0

438 rows × 6546 columns

In [249]:
```
lda_corpus_1 = LatentDirichletAllocation(n_components=4, max_iter=500,
                                         doc_topic_prior = 0.25,
                                         topic_word_prior = 0.25).fit(bow_corpus_1)
```

In [250]:
```
no_top_words = 15
display_topics(lda_corpus_1, bow_vectorizer.get_feature_names(), no_top_words)
```

```
Topic 0:
really like place want know great time little good think people take service beer pretty
Topic 1:
place good time like great really look food much people nice drink know bar little
Topic 2:
und die der ist auch nicht sehr da das ein man war mit im ich
Topic 3:
food good like place restaurant great order chicken eat time menu dish service cheese really
```

In [251]:
```
word_weights_1 = lda_corpus_1.components_ / lda_corpus_1.components_.sum(axis=1)[:, np.newaxis]
#pd.DataFrame(word_weights.T, index = bow_vectorizer.get_feature_names()).T
```

In [252]:
```
word_weights_df_1 = pd.DataFrame(word_weights_1.T, index = bow_vectorizer.get_feature_names(),
             columns = ["Topic_" + str(i) for i in range(4)])
word_weights_df_1.head(10)
```

Out[252]:

	Topic_0	Topic_1	Topic_2	Topic_3
00pm	0.000196	0.000018	0.000057	0.000021
100th	0.000196	0.000018	0.000057	0.000021
10pm	0.000352	0.000018	0.000057	0.000021
10x	0.000039	0.000088	0.000057	0.000022
11am	0.000039	0.000089	0.000057	0.000021
11pm	0.000039	0.000018	0.000273	0.000025
16th	0.000039	0.000089	0.000057	0.000021
1970s	0.000192	0.000019	0.000057	0.000022

In [261]:
```
word_weights_df_1.sort_values(by='Topic_0',ascending=False).head(10)
```

Out[261]:

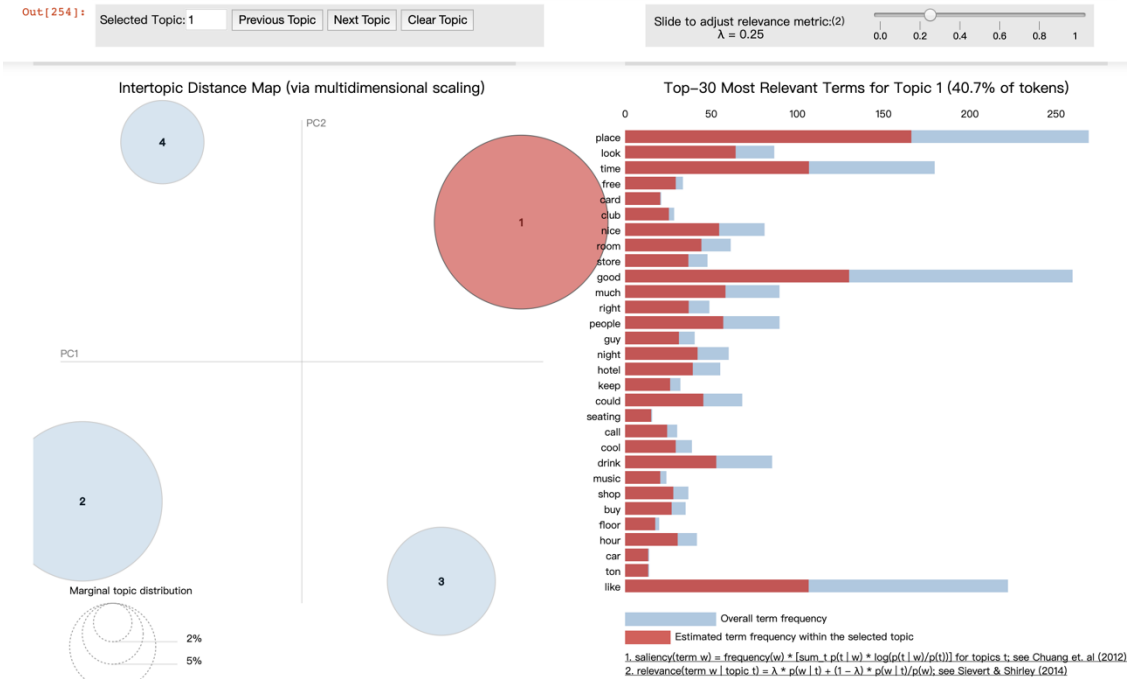
	Topic_0	Topic_1	Topic_2	Topic_3
really	0.006578	0.005432	0.000057	0.004356
like	0.005254	0.008633	0.000057	0.008707
place	0.005155	0.013462	0.000057	0.007526
want	0.005063	0.002157	0.000058	0.002669
know	0.004554	0.004121	0.000057	0.002723
great	0.004510	0.005700	0.000057	0.006048
time	0.004482	0.008646	0.000057	0.004953
little	0.004424	0.004062	0.000058	0.001699
good	0.004028	0.010535	0.000057	0.010621
think	0.003953	0.003224	0.000057	0.003423

Visualization for corpus 1

In [254]:
```
#visualization
#prepare to display result in the Jupyter notebook
pyLDAvis.enable_notebook()

#run the visualization [mds is a function to use for visualizing the "distance" between topics]
pyLDAvis.sklearn.prepare(lda_corpus_1, bow_corpus_1, bow_vectorizer, mds='tsne', R = 50)
```

Out[254]:



In [167]:
```
#find the dominant topic
lda_topic_weights_1 = lda_corpus_1.transform(bow_corpus_1)


#array of document "names" and topic "names" ("names" are just indecies)
doc_names_1 = ["Doc_" + str(i) for i in range(len(normalized_corpus_1))]
topic_names_1 = ["Topic_" + str(i) for i in range(4)]

#convert to dataframe
df_document_topic_1 = pd.DataFrame(np.round(lda_topic_weights_1, 4), columns=topic_names_1, index=doc_names_1)
df_document_topic_1.head(5)
```

Out[167]:

	Topic_0	Topic_1	Topic_2	Topic_3
Doc_0	0.9458	0.0171	0.0187	0.0184
Doc_1	0.3763	0.0405	0.5376	0.0456
Doc_2	0.9427	0.0174	0.0198	0.0201
Doc_3	0.8896	0.0274	0.0288	0.0541
Doc_4	0.9697	0.0091	0.0107	0.0105

In [168]:
```
#vector of indecies for columns with the highest value by each row in df_document_topic
dominant_topic_1 = np.argmax(df_document_topic_1.values, axis=1)

#add dominant_topic as a column to df_document_topic
df_document_topic_1['dominant_topic'] = dominant_topic_1
df_document_topic_1.head(5)
```

Out[168]:

	Topic_0	Topic_1	Topic_2	Topic_3	dominant_topic
Doc_0	0.9458	0.0171	0.0187	0.0184	0
Doc_1	0.3763	0.0405	0.5376	0.0456	2
Doc_2	0.9427	0.0174	0.0198	0.0201	0
Doc_3	0.8896	0.0274	0.0288	0.0541	0
Doc_4	0.9697	0.0091	0.0107	0.0105	0

Topic modeling evaluation for corpus 1

In [175]:
```
#topic modeling evaluation
from sklearn.datasets import fetch_20newsgroups

!{sys.executable} -m pip install gensim
import gensim

from gensim.models.coherencemodel import CoherenceModel
from gensim.corpora.dictionary import Dictionary
```

```
Requirement already satisfied: gensim in /anaconda3/lib/python3.7/site-packages (3.8.2)
Requirement already satisfied: smart-open>=1.8.1 in /anaconda3/lib/python3.7/site-packages (from gensim) (2.0.0)
Requirement already satisfied: six>=1.5.0 in /anaconda3/lib/python3.7/site-packages (from gensim) (1.12.0)
Requirement already satisfied: numpy>=1.11.3 in /anaconda3/lib/python3.7/site-packages (from gensim) (1.16.4)
Requirement already satisfied: scipy>=1.0.0 in /anaconda3/lib/python3.7/site-packages (from gensim) (1.3.1)
Requirement already satisfied: boto in /anaconda3/lib/python3.7/site-packages (from smart-open>=1.8.1->gensim) (2.49.0)
Requirement already satisfied: requests in /anaconda3/lib/python3.7/site-packages (from smart-open>=1.8.1->gensim) (2.22.0)
Requirement already satisfied: boto3 in /anaconda3/lib/python3.7/site-packages (from smart-open>=1.8.1->gensim) (1.13.1)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /anaconda3/lib/python3.7/site-packages (from requests->smart-open>=1.8
24.2)
Requirement already satisfied: idna<2.9,>=2.5 in /anaconda3/lib/python3.7/site-packages (from requests->smart-open>=1.8.1->gensim) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /anaconda3/lib/python3.7/site-packages (from requests->smart-open>=1.8.1->gensim) (3.0.4
Requirement already satisfied: certifi>=2017.4.17 in /anaconda3/lib/python3.7/site-packages (from requests->smart-open>=1.8.1->gensim) (2019.9.1
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /anaconda3/lib/python3.7/site-packages (from boto3->smart-open>=1.8.1->gensim) (0.9.5)
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /anaconda3/lib/python3.7/site-packages (from boto3->smart-open>=1.8.1->gensim) (0.3.3
Requirement already satisfied: botocore<1.17.0,>=1.16.1 in /anaconda3/lib/python3.7/site-packages (from boto3->smart-open>=1.8.1->gensim) (1.16.
Requirement already satisfied: docutils<0.16,>=0.10 in /anaconda3/lib/python3.7/site-packages (from botocore<1.17.0,>=1.16.1->boto3->smart-open>
(0.14)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /anaconda3/lib/python3.7/site-packages (from botocore<1.17.0,>=1.16.1->boto3->smar
gensim) (2.8.0)
```

In [178]:
```
#tokenizing the corpus
corpus_tokenized_1 = [tokenize_text(normalized_corpus_1[doc_id]) for doc_id in range(len(normalized_corpus_1))]

#Dictionary of the corpus:
dictionary_1 = Dictionary(corpus_tokenized_1)

#Bag-of-words representation for each document of the corpus:
corpus_bow_1 = [dictionary_1.doc2bow(doc) for doc in corpus_tokenized_1]

#top 20 words for each topic (using the function defined in session prep)
topic_topwords_1 = get_topic_words(vectorizer = bow_vectorizer, lda_model = lda_corpus_1, n_words=20)
```

In [187]:
```
cm = CoherenceModel(topics=topic_topwords_1,
                    corpus = corpus_bow_1 ,
                    dictionary = dictionary_1, coherence='u_mass')
print("Coherence score for the model: ", np.round(cm.get_coherence(), 4))  # get coherence value
print("Coherence score by topic (higher values are better): ", np.round(cm.get_coherence_per_topic(),4))
print("Log-Likelihood (higher values are better): ", lda_corpus_1.score(bow_corpus_1))
print("Perplexity (lower values are better): ", lda_corpus_1.perplexity(bow_corpus_1))
```

```
Coherence score for the model:  -8.4618
Coherence score by topic (higher values are better):  [ -1.3719  -0.7877 -16.9908 -14.6968]
Log-Likelihood (higher values are better):  -242021.92116508883
Perplexity (lower values are better):  2921.594001707484
```

For corpus 2

In [232]:
```
normalized_corpus_2 = normalize_corpus(corpus_2)

#define the bag-of-words vectorizer:
bow_vectorizer = CountVectorizer()

#vectorize the normalized data:
bow_corpus_2 = bow_vectorizer.fit_transform(normalized_corpus_2)


lda_corpus_2 = LatentDirichletAllocation(n_components=4, max_iter=500,
                                         doc_topic_prior = 0.9,
                                         topic_word_prior = 0.9).fit(bow_corpus_2)
```

In [233]:
```
lda_corpus_2 = LatentDirichletAllocation(n_components=4, max_iter=500,
                                         doc_topic_prior = 0.25,
                                         topic_word_prior = 0.25).fit(bow_corpus_2)
```

In [235]:
```
no_top_words = 15
display_topics(lda_corpus_2, bow_vectorizer.get_feature_names(), no_top_words)

word_weights_2 = lda_corpus_2.components_ / lda_corpus_2.components_.sum(axis=1)[:, np.newaxis]
#pd.DataFrame(word_weights.T, index = bow_vectorizer.get_feature_names()).T

word_weights_df_2 = pd.DataFrame(word_weights_2.T, index = bow_vectorizer.get_feature_names(),
             columns = ["Topic_" + str(i) for i in range(4)])
word_weights_df_2.head(10)
```

```
Topic 0:
good food place order like great time service really restaurant eat menu love chicken well
Topic 1:
und die de ich der da ist le war et auch nicht sehr man mit
Topic 2:
place time like great good take room look service really well people work know even
Topic 3:
pizza grocery crust store bakery market produce slice item topping deli yelp www http fresh
```

Out[235]:

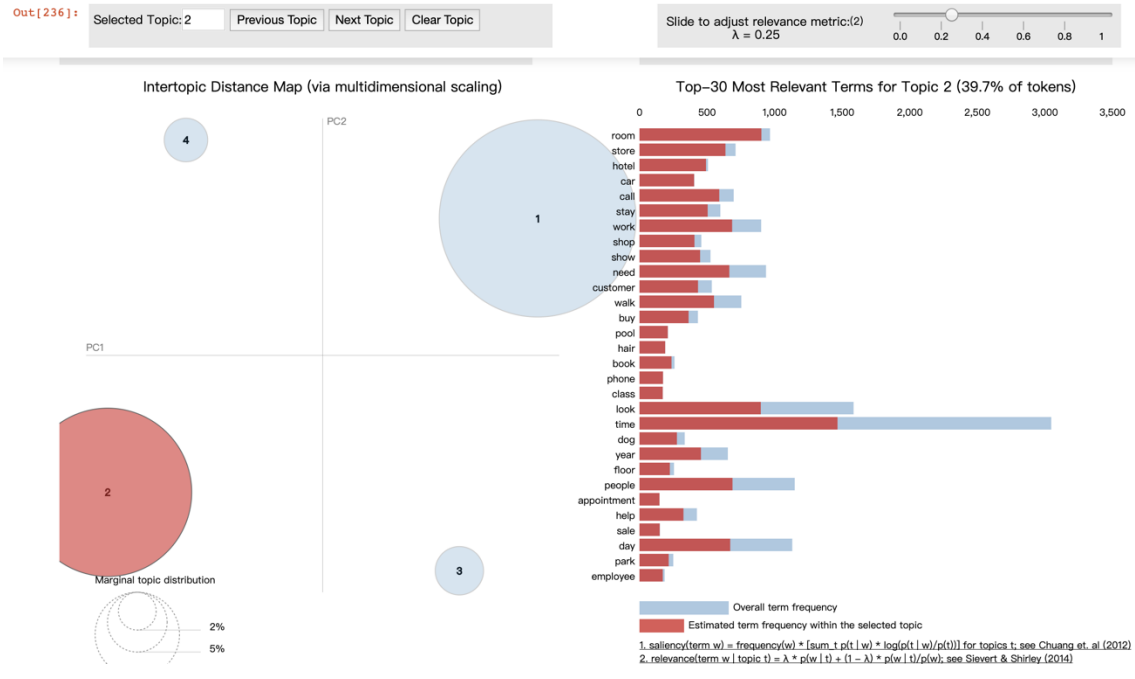
	Topic_0	Topic_1	Topic_2	Topic_3
00006p7	0.000001	0.000014	0.000007	0.000016
00am	0.000010	0.000014	0.000002	0.000016
00pm	0.000028	0.000014	0.000002	0.000249
00this	0.000005	0.000014	0.000001	0.000016
01pm	0.000001	0.000014	0.000007	0.000016
100c	0.000001	0.000014	0.000007	0.000016
100k	0.000001	0.000067	0.000002	0.000016
100m	0.000001	0.000014	0.000001	0.000080
100s	0.000001	0.000014	0.000007	0.000016
100th	0.000001	0.000014	0.000001	0.000081

Visualization for corpus 2 ¶

In [236]:
```
#visualization
#prepare to display result in the Jupyter notebook
pyLDAvis.enable_notebook()

#run the visualization [mds is a function to use for visualizing the "distance" between topics]
pyLDAvis.sklearn.prepare(lda_corpus_2, bow_corpus_2, bow_vectorizer, mds='tsne')
```

Out[236]:



In [196]:
```
#find the dominant topic
lda_topic_weights_2 = lda_corpus_2.transform(bow_corpus_2)

#array of document "names" and topic "names" ("names" are just indecies)
doc_names_2 = ["Doc_" + str(i) for i in range(len(normalized_corpus_2))]
topic_names_2 = ["Topic_" + str(i) for i in range(4)]

#convert to dataframe
df_document_topic_2 = pd.DataFrame(np.round(lda_topic_weights_2, 4), columns=topic_names_2, index=doc_names_2)
df_document_topic_2.head(5)
```

Out[196]:

	Topic_0	Topic_1	Topic_2	Topic_3
Doc_0	0.8569	0.0412	0.0629	0.0389
Doc_1	0.9733	0.0049	0.0169	0.0049
Doc_2	0.9291	0.0077	0.0564	0.0068
Doc_3	0.0850	0.0472	0.8206	0.0471
Doc_4	0.6907	0.1031	0.1142	0.0920

In [197]:
```
#vector of indecies for columns with the highest value by each row in df_document_topic
dominant_topic_2 = np.argmax(df_document_topic_2.values, axis=1)

#add dominant_topic as a column to df_document_topic
df_document_topic_2['dominant_topic'] = dominant_topic_2
df_document_topic_2.head(5)
```

Out[197]:

	Topic_0	Topic_1	Topic_2	Topic_3	dominant_topic
Doc_0	0.8569	0.0412	0.0629	0.0389	0
Doc_1	0.9733	0.0049	0.0169	0.0049	0
Doc_2	0.9291	0.0077	0.0564	0.0068	0
Doc_3	0.0850	0.0472	0.8206	0.0471	2
Doc_4	0.6907	0.1031	0.1142	0.0920	0

Topic modeling evaluation for corpus 2

In [198]:
```
#tokenizing the corpus
corpus_tokenized_2 = [tokenize_text(normalized_corpus_2[doc_id]) for doc_id in range(len(normalized_corpus_2))]

#Dictionary of the corpus:
dictionary_2 = Dictionary(corpus_tokenized_2)

#Bag-of-words representation for each document of the corpus:
corpus_bow_2 = [dictionary_2.doc2bow(doc) for doc in corpus_tokenized_2]

#top 20 words for each topic (using the function defined in session prep)
topic_topwords_2 = get_topic_words(vectorizer = bow_vectorizer, lda_model = lda_corpus_2, n_words=20)
```

In [199]:
```
cm = CoherenceModel(topics=topic_topwords_2,
                    corpus = corpus_bow_2 ,
                    dictionary = dictionary_2, coherence='u_mass')
print("Coherence score for the model: ", np.round(cm.get_coherence(), 4))  # get coherence value
print("Coherence score by topic (higher values are better): ", np.round(cm.get_coherence_per_topic(),4))
print("Log-Likelihood (higher values are better): ", lda_corpus_2.score(bow_corpus_2))
print("Perplexity (lower values are better): ", lda_corpus_2.perplexity(bow_corpus_2))
```

```
Coherence score for the model:  -1.8138
Coherence score by topic (higher values are better):  [-1.4651 -3.3716 -1.4989 -0.9196]
Log-Likelihood (higher values are better):  -3295949.632462239
Perplexity (lower values are better):  2827.9134912767513
```

For corpus 3

In [257]:
```
normalized_corpus_3 = normalize_corpus(corpus_3)

#define the bag-of-words vectorizer:
bow_vectorizer = CountVectorizer()

#vectorize the normalized data:
bow_corpus_3 = bow_vectorizer.fit_transform(normalized_corpus_3)


lda_corpus_3 = LatentDirichletAllocation(n_components=4, max_iter=500,
                                         doc_topic_prior = 0.9,
                                         topic_word_prior = 0.9).fit(bow_corpus_3)
```

In [258]:
```
no_top_words = 15
display_topics(lda_corpus_3, bow_vectorizer.get_feature_names(), no_top_words)

word_weights_3 = lda_corpus_3.components_ / lda_corpus_3.components_.sum(axis=1)[:, np.newaxis]
#pd.DataFrame(word_weights.T, index = bow_vectorizer.get_feature_names()).T

word_weights_df_3 = pd.DataFrame(word_weights_3.T, index = bow_vectorizer.get_feature_names(),
             columns = ["Topic_" + str(i) for i in range(4)])
word_weights_df_3.head(10)
```

```
Topic 0:
great place room friendly staff nice love good bar vega always like night area beer
Topic 1:
time take service work call customer give us day never need know want like even
Topic 2:
food good order place like service great restaurant time eat really chicken delicious menu well
Topic 3:
de le die la et un und pour der da est au ist que pa
```

Out[258]:

	Topic_0	Topic_1	Topic_2	Topic_3
000th	0.000013	0.000004	0.000003	0.000027
00am	0.000021	0.000017	0.000025	0.000029
00p	0.000009	0.000006	0.000003	0.000031
00pm	0.000021	0.000054	0.000006	0.000029
01am	0.000012	0.000004	0.000003	0.000027
05pm	0.000008	0.000015	0.000003	0.000028
07pm	0.000010	0.000006	0.000003	0.000029
0n	0.000007	0.000008	0.000003	0.000027
100th	0.000015	0.000005	0.000004	0.000032
100x	0.000007	0.000004	0.000006	0.000029

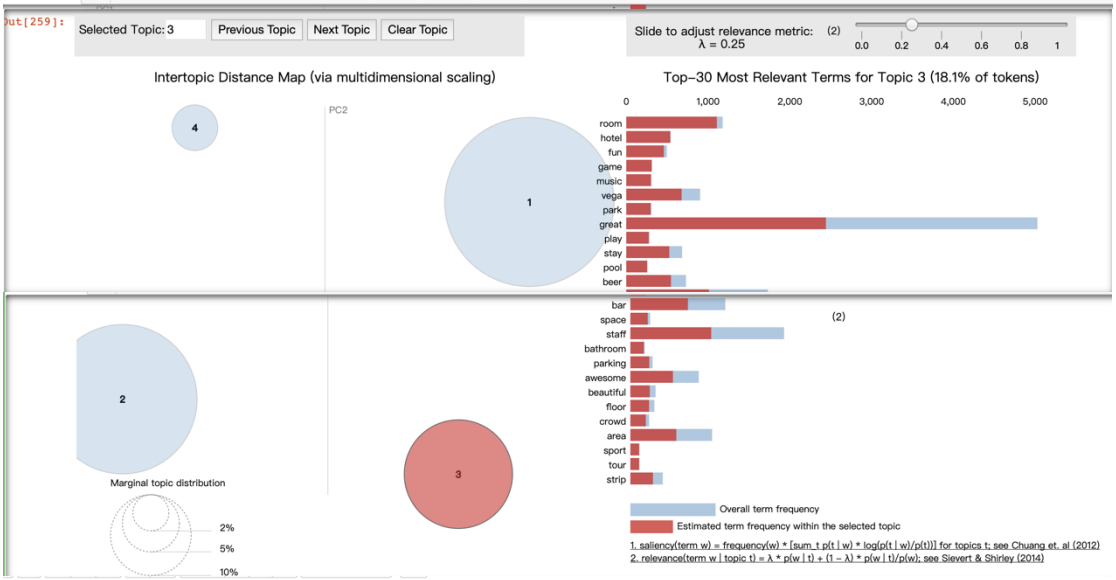
Visuaization for corpus 3

In [259]:

```
#visualization
#prepare to display result in the Jupyter notebook
pyLDAvis.enable_notebook()

#run the visualization [mds is a function to use for visualizing the "distance" between topics]
pyLDAvis.sklearn.prepare(lda_corpus_3, bow_corpus_3, bow_vectorizer, mds='tsne')
```

Out[259]:



In [203]:

```
#find the dominant topic
lda_topic_weights_3 = lda_corpus_3.transform(bow_corpus_3)


#array of document "names" and topic "names" ("names" are just indecies)
doc_names_3 = ["Doc_" + str(i) for i in range(len(normalized_corpus_3))]
topic_names_3 = ["Topic_" + str(i) for i in range(4)]

#convert to dataframe
df_document_topic_3 = pd.DataFrame(np.round(lda_topic_weights_3, 4), columns=topic_names_3, index=doc_names_3)
df_document_topic_3.head(5)
```

Out[203]:

	Topic_0	Topic_1	Topic_2	Topic_3
Doc_0	0.1141	0.0975	0.0465	0.7419
Doc_1	0.1186	0.4193	0.0178	0.4443
Doc_2	0.4517	0.3831	0.0293	0.1359
Doc_3	0.5709	0.1399	0.0450	0.2442
Doc_4	0.1846	0.6867	0.0602	0.0685

In [204]:

```
#vector of indecies for columns with the highest value by each row in df_document_topic
dominant_topic_3 = np.argmax(df_document_topic_3.values, axis=1)

#add dominant_topic as a column to df_document_topic
df_document_topic_3['dominant_topic'] = dominant_topic_3
df_document_topic_3.head(5)
```

Out[204]:

	Topic_0	Topic_1	Topic_2	Topic_3	dominant_topic
Doc_0	0.1141	0.0975	0.0465	0.7419	3
Doc_1	0.1186	0.4193	0.0178	0.4443	3
Doc_2	0.4517	0.3831	0.0293	0.1359	0
Doc_3	0.5709	0.1399	0.0450	0.2442	0
Doc_4	0.1846	0.6867	0.0602	0.0685	1

Topic modeling evaluation for corpus 3

In [205]:
```
#tokenizing the corpus
corpus_tokenized_3 = [tokenize_text(normalized_corpus_3[doc_id]) for doc_id in range(len(normalized_corpus_3))]

#Dictionary of the corpus:
dictionary_3 = Dictionary(corpus_tokenized_3)

#Bag-of-words representation for each document of the corpus:
corpus_bow_3 = [dictionary_3.doc2bow(doc) for doc in corpus_tokenized_3]

#top 20 words for each topic (using the function defined in session prep)
topic_topwords_3 = get_topic_words(vectorizer = bow_vectorizer, lda_model = lda_corpus_3, n_words=20)
```

In [206]:
```
cm = CoherenceModel(topics=topic_topwords_3,
                    corpus = corpus_bow_3 ,
                    dictionary = dictionary_3, coherence='u_mass')
print("Coherence score for the model: ", np.round(cm.get_coherence(), 4))  # get coherence value
print("Coherence score by topic (higher values are better): ", np.round(cm.get_coherence_per_topic(),4))
print("Log-Likelihood (higher values are better): ", lda_corpus_3.score(bow_corpus_3))
print("Perplexity (lower values are better): ", lda_corpus_3.perplexity(bow_corpus_3))
```

```
Coherence score for the model:  -3.4067
Coherence score by topic (higher values are better):  [-2.1475 -1.7145 -8.1221 -1.6426]
Log-Likelihood (higher values are better):  -4905774.101650478
Perplexity (lower values are better):  2434.479349146027
```

b). VADER Lexicon-Based Sentiment Analysis

In [2]:
```
import json
import pandas as pd
import numpy as np
#OPENING YELP REVIEWS DATASETS
#read the data from disk and split into lines
#we use .strip() to remove the final (empty) line
with open("yelp_academic_dataset_review.json") as f:
    reviews = f.read().strip().split("\n")

#each line of the file is a separate JSON object
reviews = [json.loads(review) for review in reviews]
print(reviews[2])
```

```
{'review_id': 'wslW2Lu4NYylb1jEapAGsw', 'user_id': 'r1NUhdNmL6yU9Bn-Yx6FTw', 'business_id': '2aFiy99vNLklCx3T_tGS9A',
'stars': 5, 'date': '2011-04-29', 'text': 'Great service! Corey is very service oriented. Works fast and very effieci
ent with his time. Going to use him again real soon to do additional IT services. thanks Corey.', 'useful': 0, 'funn
y': 0, 'cool': 0, 'type': 'review'}
```

In [4]:
```
###### extract random sample data
import random
random.seed(17)
sample = random.sample(reviews, 20000)
```

In [8]:
```
sample_reviews = pd.DataFrame(sample)
```

In [9]:
```
sample_reviews['labels'] = ['positive' if sample_reviews.loc[i,'stars']>= 4 else 'negative' for i in range(len(sample_r
```

In [35]:
```
from sklearn.model_selection import train_test_split
np.random.seed(17)
train_reviews, test_reviews = train_test_split(sample_reviews, test_size=0.2)
```

In [36]:
```
test_reviews
```

Out[36]:

	review_id	user_id	business_id	stars	date	text	useful	funny	cool	type	labels
2953	Tsd9XyFIwSd1Lxm1x-JWYQ	KatSA3PejdQkhCf0ZHUqHQ	SXnXltivyE3i3FQbydak-A	4	2009-08-05	Harbord Fish and Chips is a small brick hut th...	4	1	1	review	positive
12758	2Ztk50mH8OBluRKjhC69WQ	mpDGDd3sEwZ3o5xH0ovQcQ	hpnFcRFzs3kg-qM1tPEg3g	1	2012-04-23	Let's just preface this review by saying that ...	17	11	3	review	negative
11794	WASM0Eq0bKootfec_6h0kQ	z7GkXK4o0dOxjDzh3AWgHg	-050d_Xlor1NpCuWkbIVaQ	2	2011-07-11	First off, lemme just say that this	2	0	0	review	negative

In [37]:
```
### train extract
train_labels = []
train_corpus = []
for i in range(len(train_reviews)):
    train_labels.append(train_reviews.iloc[i]['labels'])
    train_corpus.append(train_reviews.iloc[i]['text'])
len(train_labels)
```

Out[37]:
```
16000
```

In [38]:
```
## test extraction
test_labels = []
test_corpus = []
for i in range(len(test_reviews)):
    test_labels.append(test_reviews.iloc[i]['labels'])
    test_corpus.append(test_reviews.iloc[i]['text'])
len(test_corpus)
```

Out[38]:
```
4000
```

In [16]:
```
#packages needed

import sys

!{sys.executable} -m pip install numpy
import numpy as np

!{sys.executable} -m pip install pandas
import pandas as pd

!{sys.executable} -m pip install nltk
import nltk

import warnings
warnings.simplefilter(action='ignore')

#text normalization function
%run ./Text_Normalization_Function.ipynb
```

```
Requirement already satisfied: numpy in /opt/miniconda3/lib/python3.7/site-packages (1.18.2)
Requirement already satisfied: pandas in /opt/miniconda3/lib/python3.7/site-packages (1.0.3)
Requirement already satisfied: pytz>=2017.2 in /opt/miniconda3/lib/python3.7/site-packages (from pandas) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/miniconda3/lib/python3.7/site-packages (from pandas)
(2.8.1)
Requirement already satisfied: numpy>=1.13.3 in /opt/miniconda3/lib/python3.7/site-packages (from pandas) (1.18.2)
Requirement already satisfied: six>=1.5 in /opt/miniconda3/lib/python3.7/site-packages (from python-dateutil>=2.6.1->
```

In [17]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
```

In [19]:
```
print(train_corpus[1], "\nScores:", analyzer.polarity_scores(train_corpus[1]),"\n")
```

```
I love Peppi's! I always get The Amazing Siegfried. It is a chicken club type of sandwich with bacon, mushrooms, prov
olone and american cheese. It is delish! I would give five stars but, their delivery guys is always slow and by the t
ime the food gets to me it's semi warm. They have really good fries too!
Scores: {'neg': 0.0, 'neu': 0.797, 'pos': 0.203, 'compound': 0.9103}
```

In [23]:
```
def analyze_sentiment_vader_lexicon(review, threshold = 0.1, verbose = False):
    scores = analyzer.polarity_scores(review)
    binary_sentiment = 'positive' if scores['compound'] >= threshold else 'negative'
    if verbose:
        print('VADER Polarity (Binary):', binary_sentiment)
        print('VADER Score:', round(scores['compound'], 2))
    return binary_sentiment,scores['compound']
```

In [40]:
```
###### check the test data polarity accuracy
VADER_polarity_test = [analyze_sentiment_vader_lexicon(review, threshold=0.1) for review in test_corpus]
VADER_polarity_test_df = pd.DataFrame(VADER_polarity_test, columns = ['VADER Polarity','VADER Score'])
VADER_polarity_test_df.head()
```

Out[40]:

	VADER Polarity	VADER Score
0	positive	0.4497
1	negative	-0.7488
2	positive	0.9132
3	positive	0.5455
4	positive	0.9881

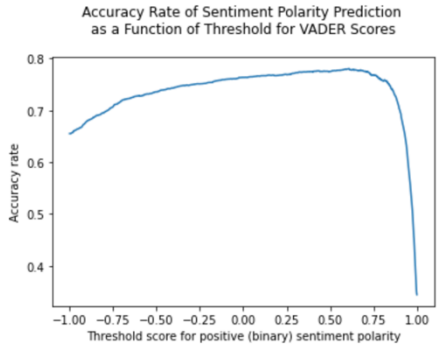
In [41]:
```
from sklearn import metrics
print('Accuracy Rate:', np.round(metrics.accuracy_score(test_labels,
                                 VADER_polarity_test_df['VADER Polarity']), 3),"\n")
```

```
Accuracy Rate: 0.767
```

In [42]:
```
########## Define a function that computes accuracy rate for different value of the threshold
def try_threshold_for_accuracy(sentiment_scores, threshold_for_pos):
    VADER_binary_polarity = ['positive' if s >= threshold_for_pos else 'negative' for s in list(sentiment_scores)]
    accuracy = metrics.accuracy_score(test_labels, VADER_binary_polarity)
    return(accuracy)
```

In [43]:
```
########## show the graph
thresholds = np.linspace(-1,1,1000)
acc_rates = [try_threshold_for_accuracy(VADER_polarity_test_df['VADER Score'],threshold) for threshold in thresholds]

plt.plot(thresholds, acc_rates)
plt.xlabel("Threshold score for positive (binary) sentiment polarity")
plt.ylabel("Accuracy rate")
plt.title("Accuracy Rate of Sentiment Polarity Prediction \n as a Function of Threshold for VADER Scores \n")
plt.show()
```



In [44]:
```
########## get the best tresholds
thresholds[acc_rates.index(max(acc_rates))]
```

Out[44]:
```
0.6096096096096095
```

In [46]:
```
########## get the accuracy rate on the best tresholds
def analyze_sentiment_vader_lexicon_2(review, threshold = 0.6096, verbose = False):
    scores = analyzer.polarity_scores(review)
    binary_sentiment = 'positive' if scores['compound'] >= threshold else 'negative'
    if verbose:
        print('VADER Polarity (Binary):', binary_sentiment)
        print('VADER Score:', round(scores['compound'], 2))
    return binary_sentiment,scores['compound']
```

In [47]:
```
###### check the test data polarity accuracy
VADER_polarity_test_2 = [analyze_sentiment_vader_lexicon_2(review, threshold=0.6096) for review in test_corpus]
VADER_polarity_test_df_2 = pd.DataFrame(VADER_polarity_test_2, columns = ['VADER Polarity','VADER Score'])
VADER_polarity_test_df_2.head()
```

Out[47]:

	VADER Polarity	VADER Score
0	negative	0.4497
1	negative	-0.7488
2	positive	0.9132
3	negative	0.5455
4	positive	0.9881

In [48]:
```
from sklearn import metrics
print('Accuracy Rate:', np.round(metrics.accuracy_score(test_labels,
                                 VADER_polarity_test_df_2['VADER Polarity']), 3),"\n")
```

```
Accuracy Rate: 0.78
```

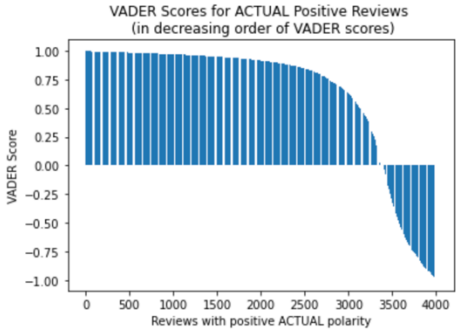
In [52]:
```
pd.crosstab(pd.Series(test_labels),
            pd.Series(VADER_polarity_test_df['VADER Polarity']),
            rownames = ['True:'],
            colnames = ['Predicted:'],
            margins = True)
```

Out[52]:

Predicted:	negative	positive	All
True:			
negative	554	825	1379
positive	108	2513	2621
All	662	3338	4000

In [49]:
```
import matplotlib.pyplot as plt

pos_reviews_scored = VADER_polarity_test_df[:4000]['VADER Score']
plt.bar(range(0, 4000), pos_reviews_scored.sort_values(ascending=False))
plt.xlabel("Reviews with positive ACTUAL polarity")
plt.ylabel("VADER Score")
plt.title("VADER Scores for ACTUAL Positive Reviews \n (in decreasing order of VADER scores)")
plt.show()
```



c) Naive Bayes Classifier-Based Sentiment Analysis

In [149]:
```
#For each star category, we would like to get equal number of samples to build model.
#So we define a function called 'getsample' to get 1000 datapoints from each star category.
sample_review=[]
sample_review2=[]
def getsample(star):
    for review in reviews:
        if review['stars']==star:
            sample_review.append(review)
    random.seed(17)
    sample=random.sample(sample_review, 1000)
    sample_review2.append(sample)
```

In [150]:
```
getsample(1)
getsample(2)
getsample(3)
getsample(4)
getsample(5)
```

In [231]:
```
#Transfer the sample datapoints into dataframe
review=[]
star=[]
for i in sample_review2:
    for j in i:
        review.append(j['text'])
        star.append(j['stars'])
df={'text':review,'stars':star}
sample_reviews=pd.DataFrame(df)
```

In [164]:
```
#Split train and test data
#We randomly select 80% of the data into train dataset and the rest 20% as test dataset.
from sklearn.model_selection import train_test_split
np.random.seed(17)
train_reviews, test_reviews = train_test_split(sample_reviews, test_size=0.2)
```

In [165]:
```
train_reviews
```

Out[165]:

	text	stars
1926	After this guy appearing on TV, I thought: "Ne...	2
805	THIS PLACE IS CLOSED \n(I passed by on 8/5/15)...	1
1674	I have been eager to try Los Sombreros for som...	2
81	Came here for the "midnight" launch of the PSV...	1
2266	We were looking for something quick to do on T...	1
...	...	...
2800	Burgers & Beers are great! If you go, ask for...	3
1337	I put 1 star only because I have to put a star...	1
406	Read reviews and had open mind. Didn't stay to...	1
2191	3.5\n\nWent to the comedy show two nights ago....	3
2671	As a gay man, I came here to support one of my...	2

4000 rows × 2 columns

In [162]:
```
import sys
import numpy as np
import pandas as pd
#!{sys.executable} -m pip install nltk
import nltk
import warnings
from sklearn import metrics
warnings.simplefilter(action='ignore')

#text normalization function
%run ./Text_Normalization_Function.ipynb
```

```
Requirement already satisfied: nltk in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (3.4.5)
Requirement already satisfied: six in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from nltk) (1.12.0)
Collecting html.parser
Requirement already satisfied: ply in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from html.parser)
(3.11)
Installing collected packages: html.parser
Successfully installed html.parser
Requirement already satisfied: pattern3 in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (3.0.0)
Requirement already satisfied: feedparser in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pattern
3) (5.2.1)
Requirement already satisfied: beautifulsoup4 in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pat
tern3) (4.8.0)
Requirement already satisfied: cherrypy in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pattern3)
(18.5.0)
Requirement already satisfied: pdfminer3k in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pattern
3) (1.3.1)
Requirement already satisfied: simplejson in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pattern
3) (3.17.0)
Requirement already satisfied: docx in c:\users\17801\appdata\local\continuum\anaconda3\lib\site-packages (from pattern3) (0.
```

In [166]:
```
test_review = np.array(test_reviews['text'])
test_polarity = np.array(test_reviews['stars'])

train_review = np.array(train_reviews['text'])
train_polarity = np.array(train_reviews['stars'])
```

In [167]:
```
normalized_test_reviews = normalize_corpus(test_review)
normalized_train_reviews = normalize_corpus(train_review)
```

In [168]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(ngram_range = (1,2))
feature_matrix_TRAIN = vectorizer.fit_transform(normalized_train_reviews).astype(float)
```

In [169]:
```
feature_matrix_TRAIN_names = vectorizer.get_feature_names()
```

In [170]:
```
feature_matrix_TRAIN_table = pd.DataFrame(data = feature_matrix_TRAIN.todense(),columns = feature_matrix_TRAIN_names)
feature_matrix_TRAIN_table.head()
```

Out[170]:

	00am	00am husband	00am starve	00ish	00ish kid	00p	00p customer	00pm	00pm apologize	00pm friday	...	zwar	zwar durch	zwar gar	zwar nicht	zwei	zwei bier	zweite	zweite chance	zwischendurch	z
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

In [171]:
```
feature_matrix_TEST = vectorizer.transform(normalized_test_reviews)
```

In [172]:
```
#Use Naive Bayes Classifer to build a multi-class model
from sklearn.naive_bayes import MultinomialNB
NBmodel = MultinomialNB()
NBmodel.fit(feature_matrix_TRAIN, train_polarity)
predict1 = NBmodel.predict_proba(feature_matrix_TEST)
```

In [173]:
```
predict1
```

Out[173]:
```
array([[0.76529458, 0.10178316, 0.0674495 , 0.05865571, 0.00681704],
       [0.512365  , 0.14271396, 0.08763701, 0.23441276, 0.02287127],
       [0.63921094, 0.13355014, 0.11128971, 0.08392706, 0.03202215],
       ...,
       [0.82274312, 0.07041189, 0.06150278, 0.03848823, 0.00685399],
       [0.87104641, 0.06502773, 0.03385891, 0.02427398, 0.00579296],
       [0.89081724, 0.07083506, 0.02319493, 0.01282154, 0.00233123]])
```

In [224]:
```
predict_bayes=[]
for i in range(len(predict1)):
    predict_list=predict1[i].tolist()
    star=predict_list.index(max(predict_list[0:4]))+1
    predict_bayes.append(star)
#print('Accuracy rate:', np.round(metrics.accuracy_score(test_polarity, predict_bayes), 3))
```

In [228]:
```
#Transfer multi-class to 2 class(positive & negative)
def startocategory(datalist):
    category=[]
    for i in datalist:
        if i>3:
            category.append(1)
        else:
            category.append(0)
    return category
```

In [229]:
```
test=startocategory(test_polarity)
predict=startocategory(predict_bayes)
```

In [230]:
```
print('Accuracy rate:', np.round(metrics.accuracy_score(test, predict), 3))
```

```
Accuracy rate: 0.804
```