SENTIMENT ANALYSIS ON YELP

IST 664
Natural Language Processing

Wengqing Ling | Dawei Guo | Huiyu Li | Liuqi Qian

Table of Contents

BACKGROUND INFORMATION

Yelp was founded in San Francisco, California in 2004. During the period of 2009 and 2014, the company expanded from the United States to Europe and Asia. The website (Yelp.com), as well as the Yelp mobile application, are the two main services it operates and both of them provide reviews and information about businesses. Currently, there are more than 135 million reviews on the businesses and restaurant reviews around the worldwide. Yelp becomes the guide to exploring the dining and relaxing places in cities.

PROBLEM STATEMENTS

For the reason that the group was located at Syracuse, NY, the following are the main goals of the project:

- Analyzing the reviews on Yelp of 50 restaurants of Syracuse to get the overall customer attitudes toward each restaurant.
- What should restaurants pay more attention to based on the customers' reviews?

DATA SOURCE

The first step of the project is to acquire. In this project, we used Yelp API to get 50 restaurants in Syracuse and use regular expressions to parse and obtain the reviews of these 50 restaurants. Finally, the code created two JSON files for us and used those two files to do sentiment analysis.

YELP API

This function is to send a request to Yelp and get a response.

```
def request(host, path, api_key, url_params=None):
    """Given your API_KEY, send a GET request to the API.
    Args:
        host (str): The domain host of the API.
        path (str): The path of the API after the domain.
        API_KEY (str): Your API Key.
        url_params (dict): An optional set of query parameters in the request.
    Returns:
        dict: The JSON response from the request.
    Raises:
        HTTPError: An error occurs from the HTTP request.
    """
    url_params = url_params or {}
    url = '{0}{1}'.format(host, quote(path.encode('utf8')))
    headers = {
        'Authorization': 'Bearer %s' % api_key,
    }

    print(u'Querying {0} ...'.format(url))

    response = requests.request('GET', url, headers=headers, params=url_params)

    return response.json()
```

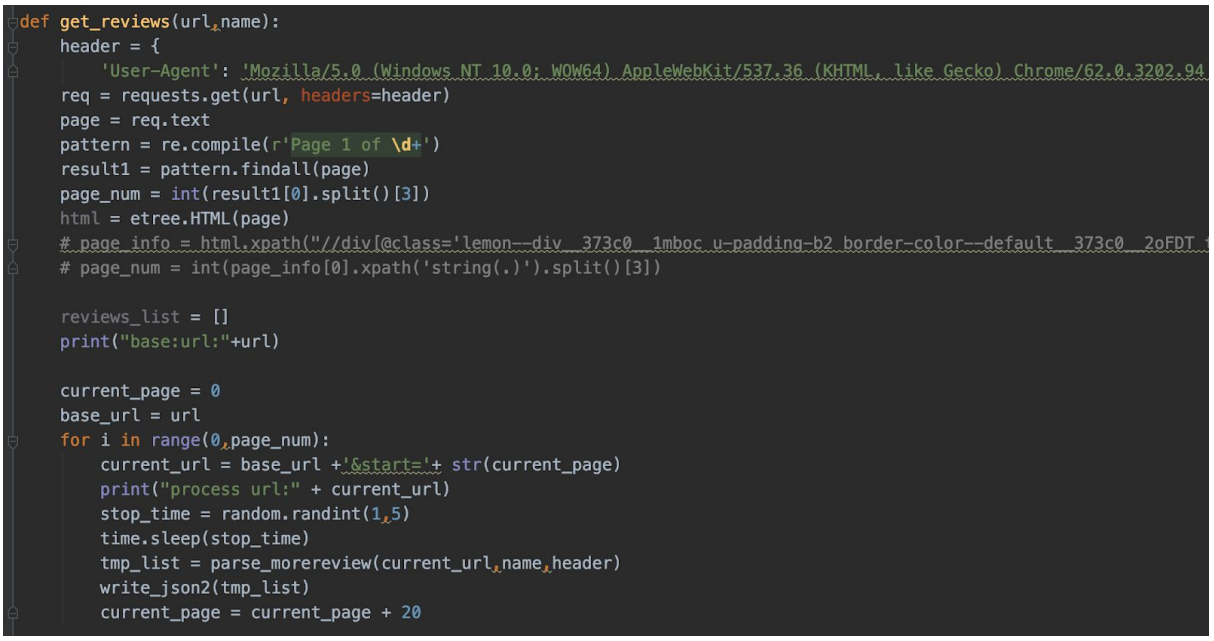
The two core functions of acquiring restaurants and reviews are query_api and get_review
query_api:
After getting response from Yelp, extract the business part, which include the basic information about the restaurants.
Then send a new requests to Yelp to obtain the review by using get_review function.

```python
def query_api(term, location):
    """Queries the API by the input values from the user.
    Args:
        term (str): The search term to query.
        location (str): The location of the business to query.
    """
    response = search(API_KEY, term, location)

    businesses = response.get('businesses')

    if not businesses:
        print(u'No businesses for {0} in {1} found.'.format(term, location))
        return

    for business in businesses:
        write_json(business)
        business_id = business['id']
        business_name = business['name']
        get_review(business_id,business_name)
        print(business)
```


```
{
    "id":"XRgHyiWi1dKW17VyyiyG_g",
    "alias":"dinosaur-bar-b-que-syracuse-3",
    "name":"Dinosaur Bar-B-Que",
    "image_url":"https://s3-media1.fl.yelpcdn.com/bphoto/brn5gkAK8SrLsIrVi458Ug/o.jpg",
    "is_closed":false,
    "url":"
https://www.yelp.com/biz/dinosaur-bar-b-que-syracuse-3?adjust_creative=FqB-
_Re8FX91LGKopdsorA&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=FqB-_Re8FX91LGKopdsorA
",
    "review_count":1801,
    "categories":[
        {
            "alias":"bbq",
            "title":"Barbeque"
        },
        {
            "alias":"tradamerican",
            "title":"American (Traditional)"
        }
    ],
    "rating":4.5,
    "coordinates":{
        "latitude":43.052489,
        "longitude":-76.154655
    },
    "transactions":[
        "pickup",
        "delivery"
    ],
    "price":"$$",
    "location":{
        "address1":"246 W Willow St",
        "address2":null,
        "address3":"",
        "city":"Syracuse",
        "zip_code":"13202",
        "country":"US",
        "state":"NY",
        "display_address":[
            "246 W Willow St",
            "Syracuse, NY 13202"
        ]
    },
    "phone":"+13154764937",
    "display_phone":"(315) 476-4937",
    "distance":1840.6775071995612
```

The screenshot above shows one record of Restaurants.json. In this project, we will use the URL and name to create Review.json. Categories used to do statistics like showing “the top 10 most popular food types in Syracuse.”

DATA SCRAPING



```
def get_reviews(url,name):
    header = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94
    req = requests.get(url, headers=header)
    page = req.text
    pattern = re.compile(r'Page 1 of \d+')
    result1 = pattern.findall(page)
    page_num = int(result1[0].split()[3])
    html = etree.HTML(page)
    # page_info = html.xpath("//div[@class='lemon--div__373c0__1mboc u-padding-b2 border-color--default__373c0__2oFDT
    # page_num = int(page_info[0].xpath('string(.)').split()[3])

    reviews_list = []
    print("base:url:"+url)

    current_page = 0
    base_url = url
    for i in range(0,page_num):
        current_url = base_url +'&start='+ str(current_page)
        print("process url:" + current_url)
        stop_time = random.randint(1,5)
        time.sleep(stop_time)
        tmp_list = parse_morereview(current_url,name,header)
        write_json2(tmp_list)
        current_page = current_page + 20
```

Get_reviews function obtains the information about the yelp website like how many pages of reviews by using regular expressions. Then by analysing the website, we found that the way that yelp change to the next review page is to change url in this “&start=” part. So if we want to crawl all reviews, we can change the “&start=” part in url by adding 20 because one page has 20 review and call parse_morereview function to send a new request to Yelp website. The reason wrote call sleep function here is to control the time that access randomly in case Yelp block us because it think our code as robot.

```python
def parse_morereview(url,name,header):
    req = requests.get(url, headers=header)
    page = req.content
    html = etree.HTML(page)

    review_list = []
    reviews = html.xpath("//p[@class='lemon--p__373c0__3Qnnj text__373c0__2pB8f comment__373c0__3EKjH text-color--normal

    if len(reviews) == 0:
        print("been blocked, please click the website to correct it")
        time.sleep(60)
        parse_morereview(url,name,header)


    ratings = html.xpath("//section[@class='lemon--section__373c0__fNwDM u-space-t4 u-padding-t4 border--top__373c0__19

    current_rating = 0

    for item in reviews:
        review = item.xpath('string(.)')
        rating = str(ratings[current_rating])
        # put all the review into list
        review_info = {}
        review_info['name'] = name
        review_info['review'] = review
        review_info['rating'] = rating
        review_list.append(review_info)
        current_rating += 1

    return review_list
```

The function **“parse_moreview”** accepts new URL and get a response by sending the request. Then use if to check if we are blocked from the Yelp website because we access it too many times. If the response is full then we use XPath to acquire ratings of reviews. After we gain all reviews and its rating, return the review_list and write it into Reviews.json.


```
{
    "name":"Dinosaur Bar-B-Que",
    "review":"Iconic BBQ location in Upstate New York. The Syracuse location is the original, alt
hough I have also been to the one in Rochester, as well as in Harlem. They all have their own dis
tinct feel, with the Rochester one feeling the "nicest" of the three, but this one was the most.
About a 20 minute wait for a table.Has a pretty good beer selection as well as some good barbecue
. Best option IMO are the ribs, and I prefer that to the pulled pork (brisket is probably in seco
nd place, but I really see Dinosaur as primarily a ribs place). Service was also quite friendly a
nd fun. Would definitely come back!",
    "rating":"5 star rating"
}
```


Here’s one record of Reviews.json. We will label the review by its rating and build to model to analyze sentiment in following research.

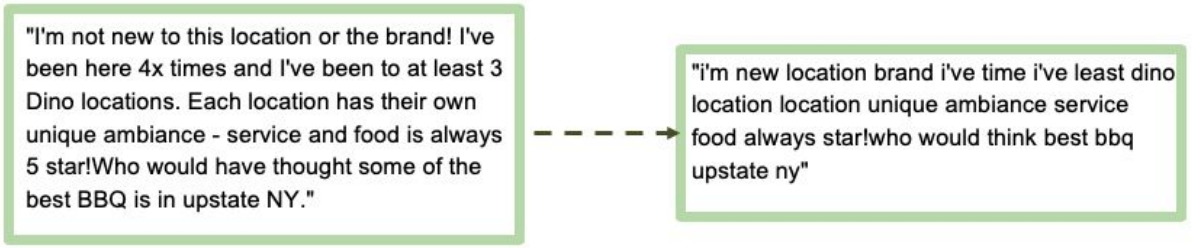
DATA PREPROCESSING

DATA CLEANING

Since we were dealing with a large set of data, the best way to clean the data was to create a function called "**clean_text**". Inside the function, we utilized several methods. First of all, we lowered all the words. Then, we tokenized the text and removed all the punctuations. Also, we erased all the invalid words that contained numbers or only had one character. Next, we cut off all the stop words based on the default stop words array from nltk.corpus. After that, all the empty tokens were deleted. What's more, we classified the words and label them by part-of-speech tagging and also lemmatized the text. For the last step, we put all the words together to be the sentences.

```
def clean_text(text):
    # lower text
    text = text.lower()
    # tokenize text and remove puncutation
    text = [word.strip(string.punctuation) for word in text.split(" ")]
    # remove words that contain numbers
    text = [word for word in text if not any(c.isdigit() for c in word)]
    # remove stop words
    stop = stopwords.words('english')
    text = [x for x in text if x not in stop]
    # remove empty tokens
    text = [t for t in text if len(t) > 0]
    # pos tag text
    pos_tags = pos_tag(text)
    # lemmatize text
    text = [WordNetLemmatizer().lemmatize(t[0], get_wordnet_pos(t[1])) for t in pos_tags]
    # remove words with only one letter
    text = [t for t in text if len(t) > 1]
    # join all
    text = " ".join(text)
    return(text)
```

The following graph shows a simple example of the original review text and the text after cleaning processing.



ADD COLUMNS

After cleaned the data, we added two more columns "**review_type**" and "**is_bad_review**" for future use. Both of the two columns were evaluated based on the review rating whose range was from 1 to 5. Reviews with the rating under 2 should be tagged as "Bad Reviews". "**Neutral Reviews**" represented the rating scores from 2 to 4. If the rating stars were greater than 4, then they will be marked as "**Good Reviews**".
However, "review_type" column defined the rating as three types: good, neutral, bad, and identified by 0,1,2. On the other hand, "is_bad_reveiw" differed only two kinds, bad and not bad. If it was bad, then tagged it to 0, otherwise, 1.

```
df['review_type'] = df['rating']
for i in range(0,len(df)):
    if float(df['rating'][i]) >= 4:
        df['review_type'][i] = "0"
    elif float(df['rating'][i]) > 2 and float(df['rating'][i]) < 4:
        df['review_type'][i] = "1"
    else:
        df['review_type'][i] = "2"


df['is_bad_review'] = df['rating']
for i in range(0,len(df)):
    if float(df['rating'][i]) > 2:
        df['is_bad_review'][i] = "0"

    else:
        df['is_bad_review'][i] = "1"
df.head()
```

DATA TRANSFORMATIONS

ADD SENTIMENT SCORES

The first step of data transformation was adding sentiment analysis features. We used Vader, which is a part of the NLTK module designed for sentiment analysis. Vader can measure which reviews are positive or negative through a lexicon of words. Four values were returned in this step:

- a neutrality score
- a positivity score
- a negativity score
- an overall score that summarizes the previous scores

COUNT LENGTH OF TEXT

Next, we added some simple metrics to count the length for every text:

- number of characters in the text
- number of words in the text

TEXT TO VECTOR

In this part, we transformed the text into numerical vectors using the word vectors (Doc2Vec). Each text has been transformed into 5 vectors. Texts with similar content will have similar representations and thus we can use those vectors as training features.

ADD TF-IDF

The final step was adding TF-IDF(Term Frequency — Inverse Document Frequency) values for every word and every review. We used TF-IDF instead of just counting how many times each word appears for each review in this step. Because those words that appear in almost every text such as 'that' would not likely bring useful information for analysis. In most cases, rare words provide more meanings.

The advantages of using TF-IDF includes:

- TF computes the classic number of times the word appears in the text
- IDF computes the relative importance of this word which depends on how many texts the word can be found

review_clean	neg	neu	pos	compound	nb_chars	nb_words	doc2vec_vector_0	doc2vec_vector_1	doc2vec_vector_2	doc2vec_vector_3	doc2vec_vector_4
iconic bbq location upstate new york syracuse ...	0.000	0.708	0.292	0.9901	598	112	-0.056116	0.014251	-0.155940	0.143853	-0.048048
recently mom boyfriend wait minute table perfe...	0.008	0.867	0.125	0.9778	1070	205	-0.063538	-0.393010	0.310593	-0.114564	-0.531521
okay say ready place ready!while visit syracus...	0.038	0.810	0.152	0.9666	930	181	-0.101185	0.044683	-0.087579	0.105258	-0.233172
first time syracuse location rochester native ...	0.013	0.785	0.202	0.9895	1027	194	0.318420	-0.109352	0.182444	-0.000059	-0.272654
town long weekend friend insist go dinosaur bb...	0.044	0.817	0.139	0.9519	849	162	0.029021	-0.146167	0.423047	0.267521	-0.147853

nb_chars	...	word_yolk	word_york	word_you	word_young	word_your	word_yum	word_yummy	word_zero	word_zing	word_zucchini
598	...	0.0	0.156028	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1070	...	0.0	0.098327	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
930	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1027	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
849	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

ANALYSIS OF SENTIMENT SCORES

We got the top five restaurants for both best reviewed and the worst-reviewed based on their average compound score. To avoid bias, we only selected restaurants with more than 30 reviews. As we can see, **Habiba's Ethiopian Kitchen** is the best restaurant in Syracuse according to the scores of reviews.

name	review	average_compound
Habiba's Ethiopian Kitchen	39	0.946664
Dang's Cafe	58	0.867238
Thai Love NY	180	0.867014
The Cider Mill	115	0.863645
Grotto	73	0.860090

Best Reviewed Restaurants

Also, we got the worst restaurant **Tang Flavour**, which got the lowest average compound score. We will give more suggestions based on this sorting after in the following part.

name	review	average_compound
Tang Flavor	108	0.583315
Red Chili	160	0.598794
Feng Cha Teahouse	32	0.607141
Alto Cinco	415	0.631960
Funk 'n Waffles	278	0.654580

Worst Reviewed Restaurants

MODELS AND INSIGHTS

RANDOM FOREST MODEL

Then we used **RandomForestClassifier** from **sklean** to build a random forest model to predict whether a review is a bad review. Also, we split our dataset into two parts: one to train our model and one to access its performances. In this model, we will use all the features we added in the last step. One thing should be noticed is that features **"review", "review_clean", "name", "review_type"** and **"is_bad_review"** shouldn't be used in this model.

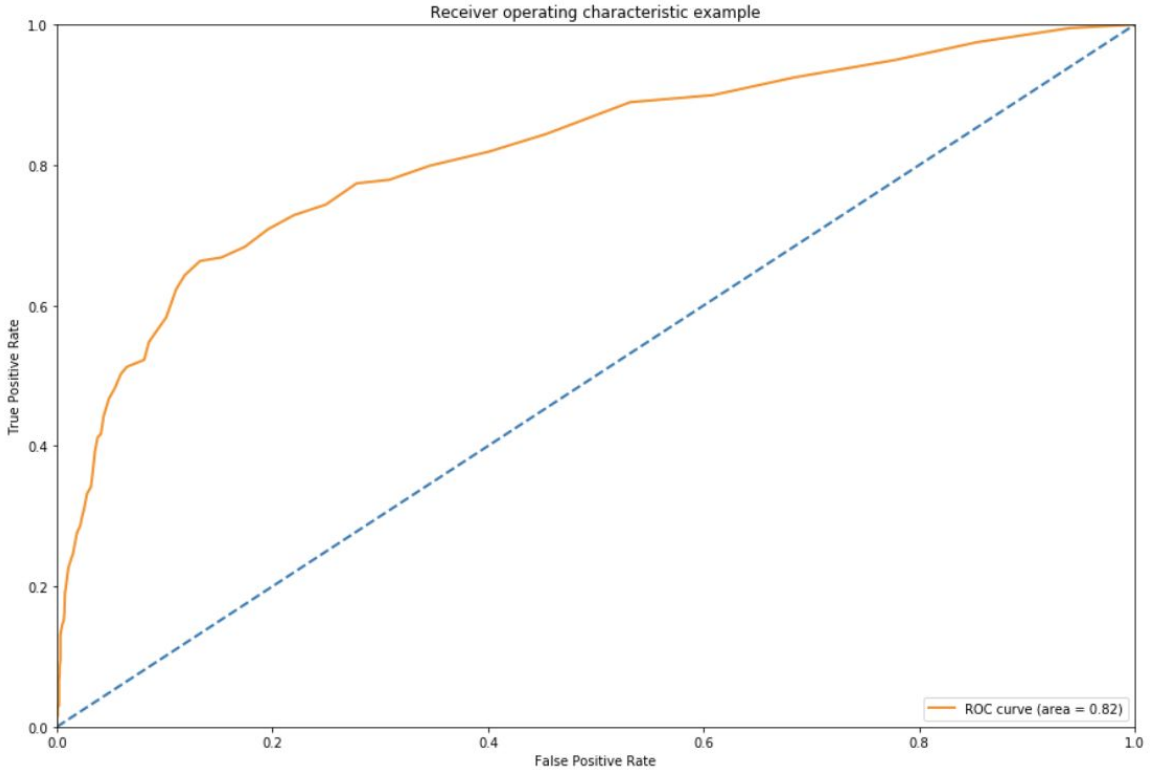
```
label = "is_bad_review"
ignore_cols = [label, "review", "review_clean","name","review_type"]
features = [c for c in reviews_df.columns if c not in ignore_cols]

# split the data into train and test
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(reviews_df[features],
                                                    test_size = 0.20, rand
```

We computed the importance of each feature and sorted them in descending order. As we expected, the compound score got the highest percentage, contributing 3% in predicting whether the review is bad. All other three sentiment scores have a pretty big influence. For all the TF-IDF, the word **"bad"** contributes more than any other words, which is a big proof of this model is working accurately.

	feature	importance
3	compound	0.036878
2	pos	0.032961
0	neg	0.028583
1	neu	0.013652
213	word_bad	0.009483
9	doc2vec_vector_3	0.009333
6	doc2vec_vector_0	0.008579
4	nb_chars	0.007974
5	nb_words	0.007447
309	word_bland	0.007217
10	doc2vec_vector_4	0.006737
8	doc2vec_vector_2	0.006566
7	doc2vec_vector_1	0.006231
1135	word_food	0.005303
1756	word_mediocre	0.004898
1272	word_great	0.004870
577	word_cold	0.004200
2862	word_terrible	0.004117
2511	word_service	0.003639
1972	word_order	0.003434

AUC-ROC CURVE

To test our model's performance, we used the AUC (Area Under The Curve) - ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. The plot shows how much the model is capable of distinguishing the target feature. All the area under the orange line is AUC. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.



Y- axis is TPR (True positive rate):

$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

X-ais is FPR (False positive rate):

$$\text{Specificity} = \frac{TN}{TN + FP}$$

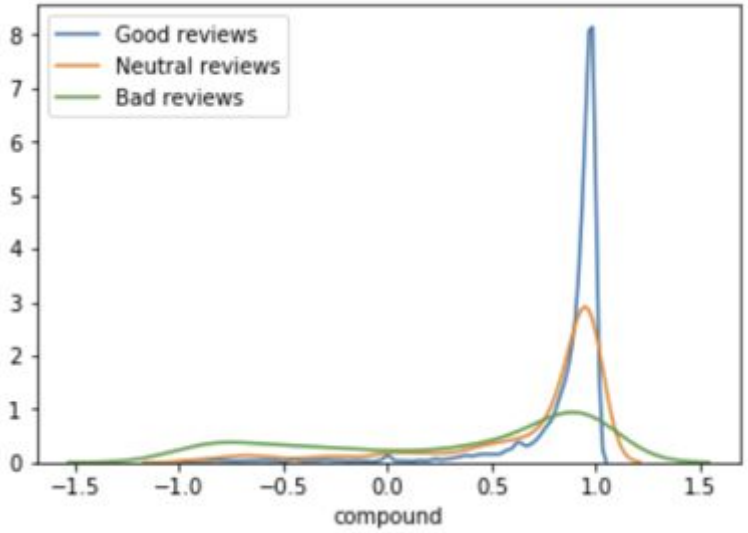
$$\text{FPR} = 1 - \text{Specificity}$$

$$= \frac{FP}{TN + FP}$$

DISTRIBUTION OF REVIEWS

We made a plot to demonstrate the distribution of each review type and its compound score. It's obvious that their distribution has a strongly different shape.

- **Good reviews**: the curve of the compound is at a peak around 1.0
- **Neutral reviews**: the curve is much flatter than good reviews
- **Bad reviews**: the curve is the flattest, and has a big proportion between -1.0 and -0.5



INFORMATION VISUALIZATION

WORD CLOUDS

(The most positive 20 comments based on all reviews from 50 restaurants)

(The most negative 20 comments based on all reviews from 50 restaurants)

CONCLUSIONS

To figure out what exactly customers are talking about the restaurants, we created two word clouds which are the most positive twenty comments based on all reviews from 50 restaurants and the most negative twenty comments based on all reviews from 50 restaurants. For the word cloud, the bigger size of a word shown on the image above, the higher frequency of the word shown in the review text. According to this regularity, we can find out what people truly care, think, and talk about their restaurants. These word clouds benefit the merchants to improve their business preferable on specific aspects.

In the positive word cloud, “great”, “menu”, “service”, and “friendly” etc are the most positive frequency words showed in the reviews, which means customers are authentic satisfied. In addition, these words are the most significant factors influencing on customer satisfactions while customers are evaluating a restaurant on Yelp.

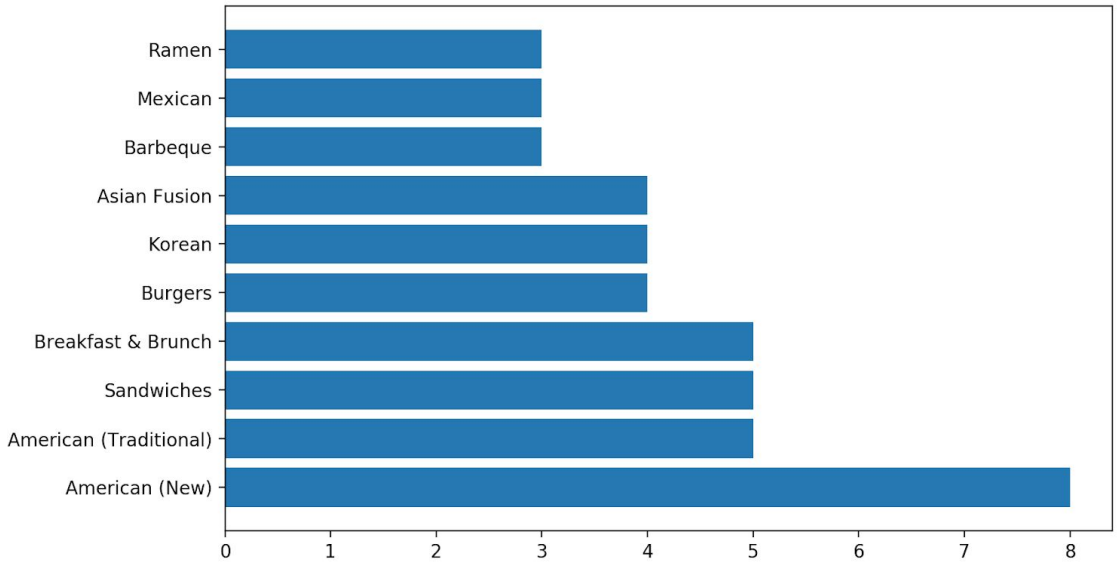
In the negative word cloud, “food”, “atmosphere”, “service”, and “taste” etc are the most negative frequency words showed in the review texts, which demonstrates that customers are authentic unsatisfied. Furthermore, these negative words are the most significant factors affecting on customer satisfactions while customers are evaluating a restaurant on Yelp.

Comparing two word clouds, we analyzed the similarity and difference between positive word cloud and negative word cloud. The similarity is they all including food and service as high frequency words. Whereas, they are slightly different while comparing them. In the positive word cloud, we can see there are 100 percent of positive words showed on the chart. Nevertheless, the negative word cloud contains not only negative words but also positive words.

RECOMMENDATIONS

Based on all the information and visualizations, we have several recommendations to the manager. First of all, merchants on Yelp should pay more attention on their food and service according to two word clouds.

Moreover, we recommend referencing the image attached below which is “the top 10 most popular food types in Syracuse” to boost a restaurants’ business. This chart illustrates that there is a total of 8 new American food restaurants located in Syracuse. Brunch, Sandwiches, and traditional american style restaurants have five each located in Syracuse. Therefore, If a merchant wants to open a restaurant focus on new american food type in Syracuse, the merchant can refer to this chart to know a variety of food types distribution in Syracuse. If a merchant wants to explore their menu with adding more food types, this chart is also helpful. All in all, it is a good reference for merchants who want to boot or expand their business in Syracuse.



Last but not least, merchants should check their negative reviews seriously, frequently, and carefully. Because negative reviews contain not only bad parts but also positive aspects of good parts. According to the output below, we find out that some customers like the food at a

restaurant, however, they don’t like the environment or location, etc. Additionally, some reviews are judging customer service but praise their particular food. After merchants checked the negative reviews on Yelp, they can know what they are doing good and bad. Finally, they can make changes and adjustments to run their business better than before.

	review	review_clean	neg
4376	Nothing better!! Love this place. Fish tacos a...	nothing well love place fish taco amazing	0.415
114	No resteraunt has the meet but hear don't go i...	resteraunt meet hear go like great atmosphere ...	0.409
6575	HORRIBLE customer service and average tasting ...	horrible customer service average tasting over...	0.377
8393	Ok this waitress is the most impatient bitch I...	ok waitress impatient bitch ever see sure empl...	0.376
595	Terrible so terrible please stay away unless y...	terrible terrible please stay away unless want...	0.351
3828	Bad service ok food bad beer lots of wine by t...	bad service ok food bad beer lot wine glass roach	0.347
3469	this is the worst place ive ever been to. the ...	bad place ive ever absolute bad service even l...	0.340
1196	Great food bad service. Small venue expect tic...	great food bad service small venue expect tick...	0.329
5814	The food is absolutely amazing but I'm not a h...	food absolutely amazing i'm huge fan staff sta...	0.323
2879	I dont care about the food tastes but this pla...	dont care food taste place's service hell wait...	0.321

PYTHON CODE

Our project code uploaded on the blackboard with file named “final project.py”.