CMPE-255 Spring 2019 Final Project Report

Yelp Restaurant Review Clustering System

Team 9: Xiaoting Jin (013842192) Siyun Liang (013708227) Chunwei Lee (012531584)

I. Introduction

As one of the largest business directory websites in the world, Yelp connects people with local business services with a monthly average of over 33 million unique visitors and Yelpers have written more than 177 million reviews by the end of Q4 2018[1]. The crowd-sourced reviews are significantly important to Yelpers during their browsing and decision-making process. However, the abundance of too many reviews makes information digestions a time-consuming process. According to Yelp, they use automated software to recommend the most helpful and reliable reviews[2], but are there any highlights across the reviews? Could we get a quick bird's-eye view of all the reviews at a glance? Our team is proposing a new feature that clusters the reviews of a business and generates a list of tags that can organize and summarize the highlights of reviews, which could help users save time and extract useful information more efficiently.

Currently, Yelp's review system supports searching via highlighted keywords extracted from reviews, such as 'Bread', 'Chicken', 'Beef' etc, but multiple-words tags are not supported. For example, in the Amazon mobile app, there's a feature called "Read reviews that mention" that enables users to see which words are the most frequently used and tap a button could straightly navigate to a filtered selection of reviews that include the term.

II. DataSet

In this project, we used two datasets from Yelp Dataset .json files which are:

1. Business.json: contains information about the store, such as business_id, name, address, stars, categories, etc.
   Size:138.3 MB; Samples:192,609; Features:27
2. Review.json: contains information of every single review, such as user_id that wrote the review, business_id that the review is written to, review content, etc.
   Size:4.71 GB; Samples:6,685,900; Features:9

The histogram of restaurant reviews shows a long tail distribution, where review counts less than 100 has a large number of occurrences, although the maximum number of reviews is 8,570, the mean and median are only 71 and 21.

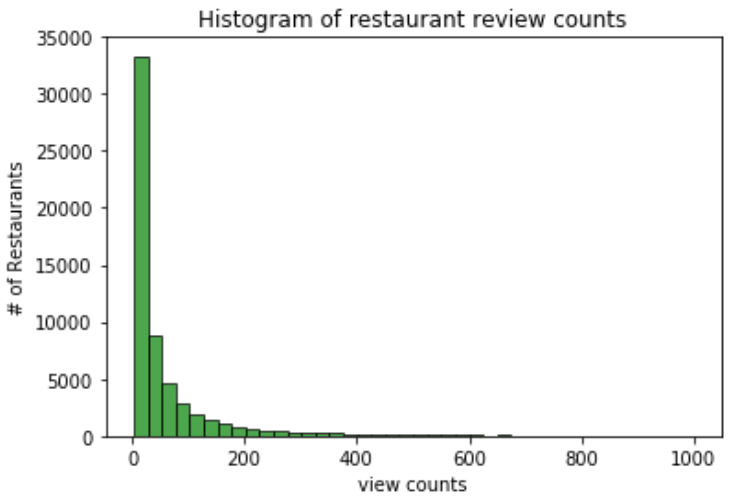

Figure 1. Histogram of restaurant review counts

III. Data Preprocessing

1. Data Loading

Our objective is to produce comments tags of each business in restaurant category, so before we start building our model, we need to collect reviews of the same restaurant at first. To accomplish that, first we load data from Business.json file, and since this .json file is nested, we cannot directly apply method from Pandas, so we build a function to store the data and use Pandas for further normalizing the data. Once we get the business data, we can start extracting each business_id that belongs to "Restaurant" category, and from the dataset, we collect a total number of **59,371** samples of business_id that are in the restaurant area. After that, we begin loading Review.json file, and because of its large size, we cannot use pandas_read_json to read the file as well, so we use the same function above to store the data. After we have these business_id and reviews data, we begin to merge comments that are from the same business_id, so that we get every comment of the store. Once we complete these works, we can start our text-preprocessing and model-building process.

2.Text Preprocessing

After merging data from these two files, we get **59,371** restaurants and **4,201,684** reviews from these restaurants, there are no NaN values in 'text' field. At this step, we just transform words into lowercase and remove a set of special symbols [!"#$%&'()*+./:<=>?@[\]^_`{|}~]:.

IV. System Design and Implementation details

Approach 1: Word Embeddings

In this approach, we use sentences to train Word2Vec model and get vectors of every single word first, then use some Parts of Speech Tagging rules to extract candidate phrases, such

as adjective + noun, verb + adverb, adverb + adjective. Together with word embeddings, we could calculate the similarity matrix across candidate tags, finally, clustering algorithms such as DBSCAN, K-Medoids and Affinity Propagation are applied to get outputs.

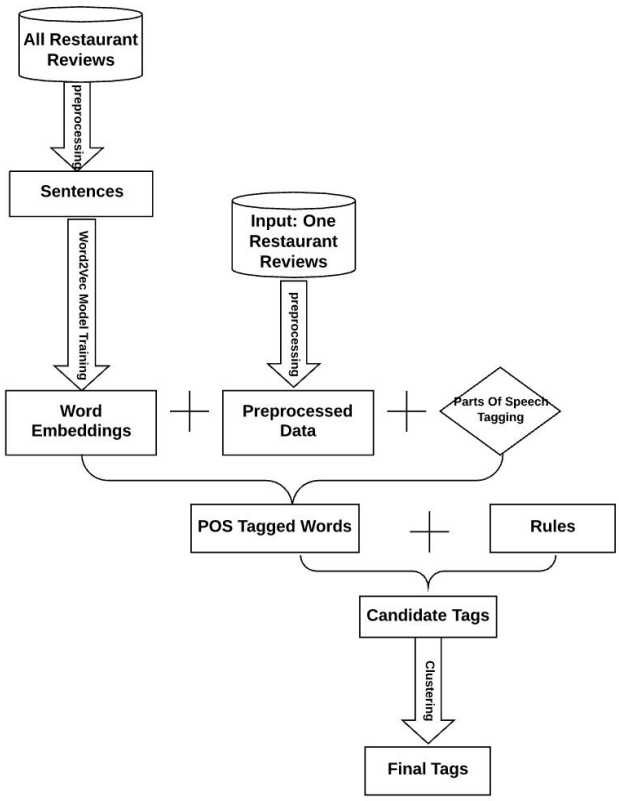

Figure 2. Flow of Approach with Word Embedding

Approach 2: Phrase Embeddings

In this approach, we generate candidate phrases by splitting sentences with punctuations, stop words and length limitations. After extracting all the candidate phrases from the dataset, we use Word2Vec from *gensim* library to get phrase embeddings, with these representative vectors, we could calculate the similarity matrix among candidate tags of each input restaurant.

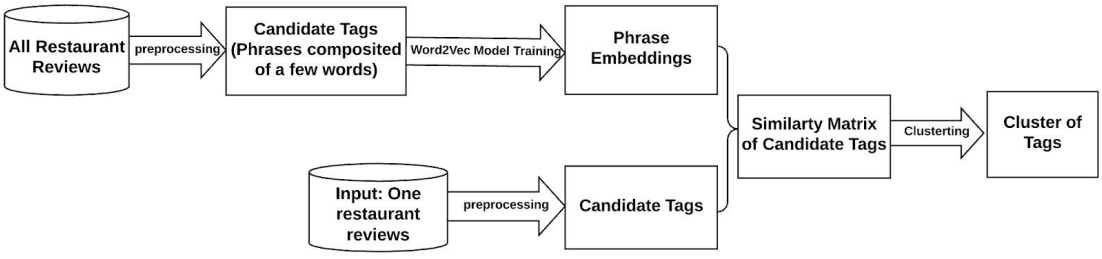

Figure 3. Flow of Approach with Phrase Embedding

Approach 3: Using PMI as metric

In this approach, Pointwise mutual information (PMI) is computed for all the bigrams and trigrams of the reviews for a specific restaurants, and top n of them are chosen as tags.

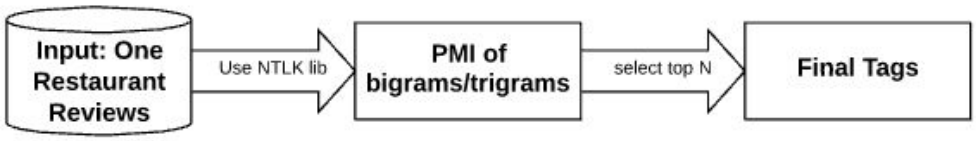

Figure 4. Flow of Approach using PMI

1. **Extract candidate phrases from sentences**

There are two methods used in generating tags candidate, which are stop words leveraging and utilization of POS tagging & rules.

A) Leveraging stop words.

To generate meaningful phrases(more than two words), stop words are used to split sentence. For example, for the review "great food at a reasonable price with nice service and location", since “at”, ”with” and “and” are stopwords, “great food”, “reasonable price” and “nice service” can be extracted from the sentence.

The quality of phrases extracted depends heavily on dictionaries of stopwords. Two popular stopwords libraries are considered here: one from Spacy(n=153) and the other is from nltk(n=307). Since the nltk one is conservative compared to the Spacy one, stopwords from Spacy is used to split phrases here. Also, a few words are appended based on our needs. For instance, “like” is added to the stopwords, since it’s usually used in the review in “I like something” or “feel like something” patterns, and this “something” is naturally the feature of this restaurant.

Figure 5. Word cloud for extracted phrases

B) Using POS Tagging and Rules

The part of speech explains a word's function and position in a sentence. There are eight main parts of speech - nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions, and interjections. The nouns, verbs, adjective are passing relatively more information in a tag. Observing the review tabs in Amazon mobile app, we make some rules of forming a tag, a few examples are:

Rules	Examples
adj + noun or noun + adj	fantastic menu, poor quality, dirtiest places, prices reasonable
adv + verb or verb + adv	highly recommend, sat immediately
auxiliary + verb	must try
noun + noun	garlic bread, bottle wine

Table I. POS Tagging Rules

Notice that the above rules are not comprehensive and not always working, so refining POS rules is an area that we could put more efforts on in future work.

2. Embeddings

A pre-trained word embedding model, word2vec is employed in the study. To calculate the similarity between candidates, Continuous Bag-of-Words (CBOW) model is chosen. Three parameters are set in the model: dimensionality, context window, and minimum count. Dimensionality is related to the quality of word embedding, usually the higher the dimensionality is the better the quality. But it adds computation time as well. Consider this project is limited to restaurant, 200 is set for dimensionality. Since most of the reviews are just a few short sentence, context window is set to 10. Minimum count is set to 10 to filter some "nonsense" phrases.

As mentioned before, we have tried 2 approaches, the first one is to input sentences and get word embeddings, the second one is to input phrases list and get phrase embeddings.

```
model.wv.most_similar("awesome")
```

```
[('fantastic', 0.9057341814041138),
 ('amazing', 0.9008509516716003),
 ('great', 0.874151349067688),
 ('terrific', 0.865390419960022),
 ('phenomenal', 0.8464846611022949),
 ('fabulous', 0.8463567495346069),
 ('incredible', 0.8384655714035034),
 ('wonderful', 0.7946398258209229),
 ('excellent', 0.7901973724365234),
 ('outstanding', 0.7634873986244202)]
```

```
model.wv.most_similar("rude")
```

```
[('disrespectful', 0.8462716341018677),
 ('unfriendly', 0.8419162034988403),
 ('unprofessional', 0.8338981866836548),
 ('condescending', 0.8123921155929565),
 ('impolite', 0.802875280380249),
 ('bitchy', 0.7854247093200684),
 ('snotty', 0.7836031317710876),
 ('dismissive', 0.7707229256629944),
 ('argumentative', 0.7523098587989807),
 ('unhelpful', 0.7446948885917664)]
```

```
model.wv.most_similar('pretty_good')
```

```
[('pretty_yummy', 0.5680291652679443),
 ('tasted_pretty_good', 0.534600555896759),
 ('pretty_salty', 0.5254586935043335),
 ('pretty_tender', 0.5237904787063599),
 ('kinda_bland', 0.49979981780052185),
 ('pretty_delicious', 0.499198853969574),
 ('actually_pretty_tasty', 0.49347805976867676),
 ('pretty_good_experience', 0.49293118715286255),
 ('pretty_flavorful', 0.4895469546318054),
 ('actually_pretty_good', 0.4829956591129303)]
```

```
model.wv.most_similar('great_place')
```

```
[('goodgreat_place', 0.48661351203918457),
 ('ideal_place', 0.4756534993648529),
 ('excellent_spot', 0.4668395221233368),
 ('placegreat_place', 0.4623957872390747),
 ('awesome_place', 0.45347321033477783),
 ('perfect_place', 0.44771289825439453),
 ('excellentgreat_place', 0.4466698169708252),
 ('friendlygreat_place', 0.44407418370246887),
 ('friendly_staffgreat_place', 0.4381403625011444),
 ('excellent_place', 0.42520672082901)]
```

Figure 6. Result of Word Embeddings and Phrase Embeddings

3. Enhance Clustering Model

To get the most representative tags for each restaurant, clustering is employed for all the reviews with respect to a specific restaurant. 3 approaches are explored here.

1) DBSCAN

DBSCAN for sklearn library is applied. Since number of reviews for different restaurant varies significantly, min_samples are set separately according to the number of reviews. Silhouette Coefficient is used to tuned parameters, but its relation with tags quality is not clear.

After clustering phrases generating from reviews, core points are used as tag candidates. For core points from the same group, majority vote strategy is applied to ensure tags are representative while not similar to each other.

```
print(get_predict('ikCg8xy5JIg_NGPx-MSIDA')) ##review number=15
```

```
Silhouette Coefficient: 0.175
['dirtiest_places', 'health_inspections', 'rude_staff', 'kensington_location']
```

```
print(get_predict('zvO-PJCpNk4fgAVUnExYAA'))  ##review number=29
```

```
Silhouette Coefficient: 0.195
['sports_grill', 'sports_bars', 'pool_tables', 'playoff_games']
```

```
print(get_predict('ERnG-1q3igX3VSgm5uLZ6A')) ##review number=74
```

```
Silhouette Coefficient: -0.092
['pizza_pazzi', 'favourite_restaurant', 'tomato_sauce', 'favourite_dish', 'order_pasta', 'great_service', 'balsamic_
vinaigrette', 'second_time', 'reasonably_priced', 'rice_balls']
```

```
print(get_predict("eU_713ec6fTGNO4BegRaww")) ##review number=135
```

```
Silhouette Coefficient: -0.102
['crab_tortellini', 'highly_recommend', 'best_italian_food', 'large_party', 'garlic_bread']
```

```
print(get_predict('KR2kRmHnRCaNzOUEGoB25w')) ##review number=279
```

```
Silhouette Coefficient: -0.119
['lola_fries', 'crocker_park', 'dont_know', 'pulled_pork', 'onion_rings', 'happy_hour', 'seated_right_away', 'rosema
ry_fries', 'veggie_burgers']
```

```
print(get_predict('Bf2fuqWbHd3L-X69FSMvmg'))##review number=313
```

```
Silhouette Coefficient: 0.107
['kensington_market', 'mexican_food', 'fish_tacos', 'pico_gallo', 'fish_taco', 'chicken_tacos']
```

Figure 7. Cluster for different number of reviews items

2) K-Medoid

In K-Medoid method, it requires a cluster number as input parameters, but, unfortunately, we only have a distance matrix that contains the distance between each phrases, so in this method, we directly use the cluster number generated by DBSCAN method as one our input, and the other input is our distance matrix. At the beginning, it randomly assigns K points as medoids, and then assigns the data points to its closest medoids, and recompute new medoids points within the cluster. The tags generated by this method is from the medoid points, and therefore the results might be different for the same business due to the randomly-chosen mechanism of first K medoids.

3) Affinity Propagation

In Affinity Propagation clustering, unlike K-Medoid method, we do not need the cluster number for input parameter, and it could also directly take distance matrix as input to do the clustering. After finishing the clustering, we have the labels, cluster number, cluster centers, and we take these cluster centers as cores, and employ the same procedure as DBSCAN to get the most frequent phrases which will becomes the tags result.

4. Using PMI

Instead of clustering reviews with word embedding techniques, there is a way to generate tags directly from the reviews of each restaurants: calculate Pointwise mutual information (PMI) for all the bigrams and trigrams in the reviews of a specific restaurants, then choose top n of them as tags.NLTK[3] has nice algorithms to support this approach. Nevertheless, this approach might produce some tags with similar meanings.

```
quickversion('ikCg8xy5JIg_NGPx-MSIDA')##redelicview number=15
```

```
['lunch_steak_sandwich', 'past_health_inspection']
```

```
quickversion('ERnG-1q3igX3VSgm5uLZ6A')  ##review number=74
```

```
['fried_risotto_ball', 'authentic_neapolitan_pizza', 'tomato_sauce', 'authentic_italian', 'italian_food', 'pizza_paz
zi']
```

```
quickversion("eU_713ec6fTGNO4BegRaww")  ##review number=135
```

```
['pepper_cream_sauce', 'shrimp_crab_tortellini', 'italian_wedding_soup', 'best_italian_food', 'cooking_class', 'high
ly_recommend', 'garlic_bread', 'tavola_italiana', 'large_group']
```

```
quickversion('Bf2fuqWbHd3L-X69FSMvmg')##review number=313
```

```
['authentic_mexican_food', 'pico_gallo', 'kensington_market', 'corn_tortilla', 'fish_taco', 'mexican_restaurant', 't
aco_pastor']
```

```
quickversion('NyLYY8q1-H3hfsTwuwLPCg') ##review number=547
```

```
['hand_washing_station', 'chicken_tikka_masala', 'tikka_masala_bowl', 'tikka_masala_sauce', 'chipotle_indian_food',
'indian_fast_food', 'highly_recommend', 'lamb_meatball', 'mango_lassi', 'samosa_chaat', 'fast_casual', 'wheat_naan']
```

```
quickversion('8mIrX_LrOnAqWsB5JrOojQ') ##review number=1289
```

```
['super_mario_bros', 'pinball_hall_fame', 'row_row_pinball', 'school_arcade_game', 'classic_arcade_game', 'row_pinba
ll_machine', 'salvation_army', 'donkey_kong', 'highly_recommend', 'memory_lane', 'star_war', 'star_trek', 'go_charit
y']
```

Figure 8. Tags Result of NLTK Algorithms

V. Experimental Results and Evaluation

In the evaluation section, we use Silhouette Coefficient to evaluate the performance of clustering. Besides, we'd like to assess our three clustering methods from two aspects which are "Diversity" and "Frequency of Occurrence" . To examine the tags diversity within one business, we first compute the cosine similarity between each tag and accumulate these similarity, then divide it by the combination C(n,2), where n is the number of the tags. (Note that the cosine similarity values are given by the word2vec model.) The result of the total similarity divided by its combination number represents the average similarity value between each tag, and the lower the value is, the more diverse the tags are, vice versa. The second evaluation criteria is the frequency of each tag that occurs in the entire comments, and to compute average tags frequency, we aggregate the times that each tag occurs in the comments, and then divides it to the number of tags to get the average occurrence time. The value of average occurrence time indicates that how many times of each tags occurs in the reviews in average, and if the value is high, it implies the the tags are more representative.

With these two evaluation criteria, we begin to test our three clustering approaches. We choose 50 & 300 randomly business ID, and as described above, within one business, we first compute the similarity (per pair) between each tags and count their average occurrence times, then repeat the same process over 50 & 300 random business, and compute their average similarity and frequency. As Figure 9-1 shows, with 50 iteration, we can observe that the average similarity of DBSCAN is 40% higher than Medoid technique and 3.5 times of Affinity Propagation method; however, the frequency of DBSCAN approach is 2.5 times of Medoid manner and 20 times of Affinity Propagation method. The same attribute can be seen in Figure 9-2 which traverses 300 random business ID and Figure 9-3 adds one new evaluating criteria which is Silhouette Coefficient Score, and although the mean similarity of DBSCAN approach is higher than other two methods, the similarity value still remain around 0.1~0.2 which is acceptable in the point of diversity; on the other hand, the occurrence time and the S.C. score of DBSCAN technique is comparily higher than other two approaches, and therefore implies a higher representativeness among user comments.

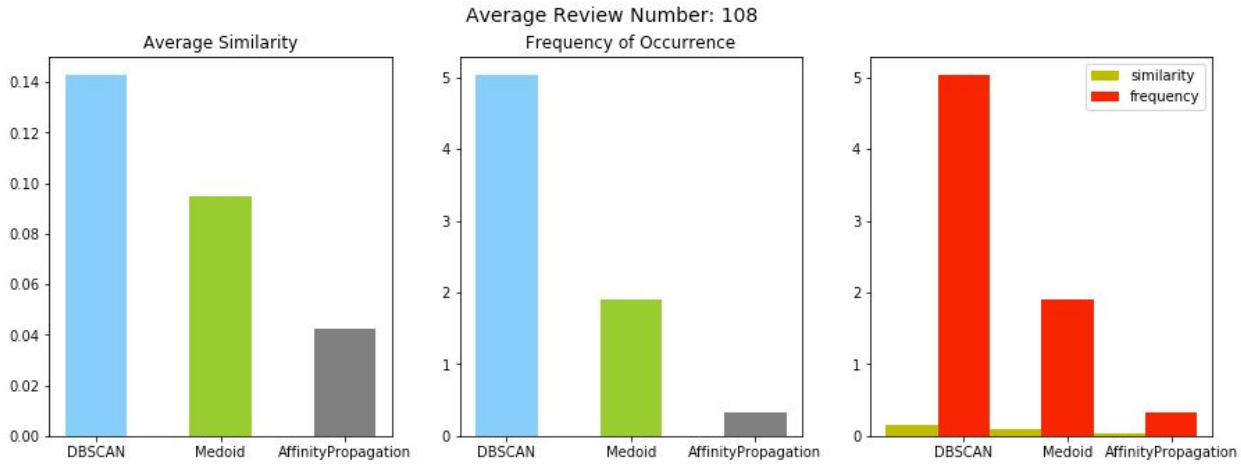

Figure 9-1. Fifty Randomly Selected Business ID

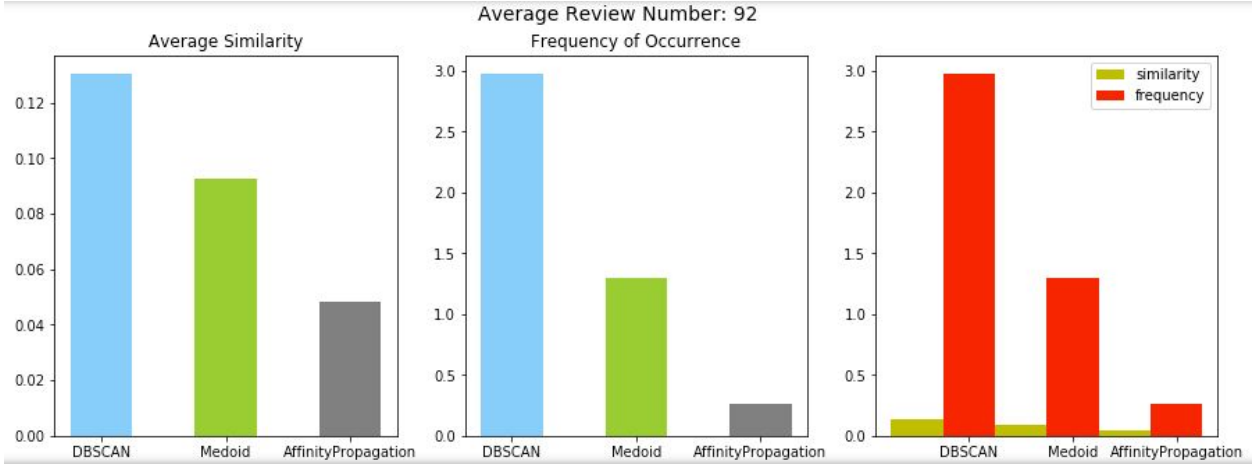

Figure 9-2. Three Hundred Randomly Selected Business ID

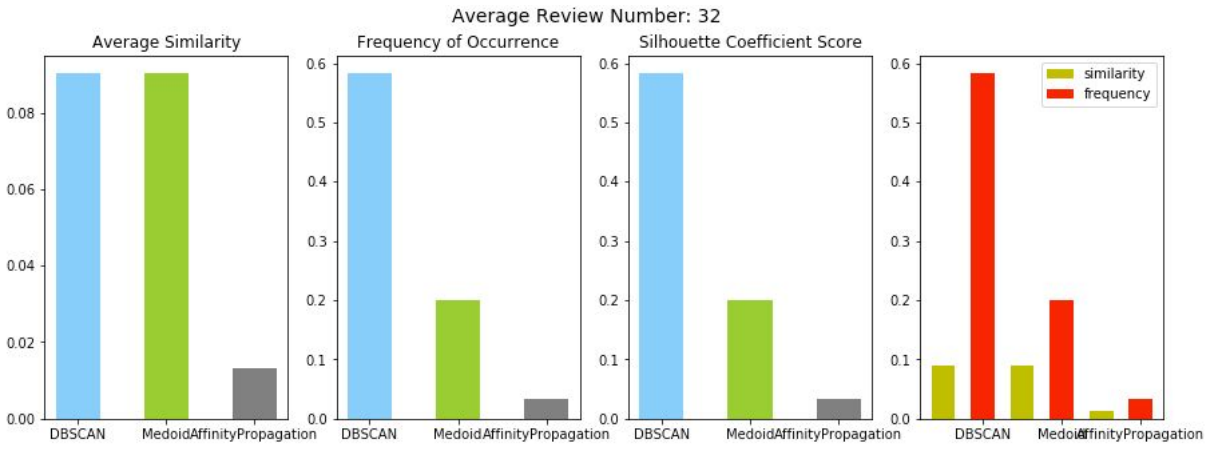

Figure 9-3. Fifty Randomly Selected Business ID

VI. Discussion and Conclusions

In this project, we are trying to build a clustering system that can generate tags among user comments. The difficulty of achieving this purpose comes from evaluation section, since it's an unsupervised learning field, and it's difficult to find a convincing and "machine-generated" metric for this kind of tasks. Most previous studies are evaluated by user testing. A few of them having access to the products launched A/B online test to see the actual effects of the model. Therefore, the lack of evaluation makes model tuning and selection difficult and somehow arbitrary.

The other problem is generating representative and easy-to-understand tags. Three approaches are experimented in the study, but each of them has its limitation. For the word embedding approach, we would miss some meaningful tags due to the difficulty of covering all the rules that are not always work for special combinations. As to the phrase embedding approach, this approach is most likely to produce nonsense tags compared to other approach because the complex structure of natural language and the multiple implication of stopwords.Also, it is possible to lose meaningful data as well since it filter out the single word between stop words. For the PMI approach, this is the most simple and efficient one of them. Since it only counts for the mutual information of the data and neglect the similarity between them, this approach is likely to generate similar tags.

VII. Task Distribution

Task	**Team**
Discussion and Proposal	All
Implementation	Siyun Liang: Approach 2&3 Xiaoting Jin: Approach 1&K-Means Chunwei Lee: Approach 2 & Evaluations
Report and Slides	All

IX. References

[1] https://www.yelp.com/about
[2] https://www.yelp-support.com/article/What-is-Yelp-s-recommendation-software?l=en_US
[3] https://www.ideals.illinois.edu/bitstream/handle/2142/18702/survey_opinionSummarization.pdf
[4] http://www.nltk.org/howto/collocations.html
[5] https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb
[6] https://blog.csdn.net/shijing_0214/article/details/71036808