Capstone Two

NLP RESTAURANT SEARCH ENGINE
PROVIDING ACCURATE RESTAURANT LISTINGS

Steve Kim

12/08/2020

Report - NLP Restaurant Search Engine

1. Introduction

1.1 Problem

Search engines are everywhere such as in e-commerce, social media, education, and general sites and they are frequently used daily by almost everyone around the world. Although restaurant search engines aren't unique and exist already in Yelp and other business review sites; I always wondered how search engines are built and processed. This project is my take on populating accurate restaurant listings using multiple datasets through natural language processing which I aim to provide specific results based on user's search input whether it's general search term (i.e. 'I want Chinese food') or more specific (i.e. 'guacamole with picadillo).

1.2 Target Audience

The model is geared towards food enthusiasts and travelers looking for specific taste and atmosphere, however it can be also useful for the general population looking for places to dine or take-out.

1.3 Dataset

In this project, I am using multiple datasets relating to restaurants, food/ingredients, and menu items. Yelp's dataset is used for getting businesses, reviews, and tips datasets which contain a wealth of restaurants' text information. Yummly's dataset is used for referencing each cuisine's ingredient list. Lastly, I web-scraped restaurant menu data from Allmenus, which contains valuable food/drink menu information. Below are the summary for each datasets:

Yelp Dataset

- 8,021,122 reviews
- 209,393 businesses
- 1,320,710 tips
- Over 1.4 million business attributes.

Yummly Dataset

- 20 cuisines (Korean, Spanish, etc.)
- 36,568 ingredients

Allmenus

- 12,000+ restaurants
- 1,000,000+ menus

2. Data Preprocessing

Prior to the natural language processing step, text datasets needed to be preprocessed such as tokenization, lemmatization, and transforming to word vectors etc. I cleaned Yelp, Yummly, and Allmenus dataset and filtered information based on restaurant names appearing in all three datasets. For instance, not all restaurants listed in Yelp dataset appeared in Allmenus dataset (restaurant menus) and not all cuisines listed on Yelp dataset is available in Yummly dataset. Therefore all aforementioned datasets require some filtering before moving forward with NLP.

2.1 Yummly Dataset

Yummly dataset did not contain any NaNs or missing values, it contained a cuisine column and ingredient column.

- It has 20 unique cuisines in which Italian, Mexican, American, Indian, and Chinese cuisines accounted for having most information.
- Yummly had 39,774 ingredient entries representing cuisine.

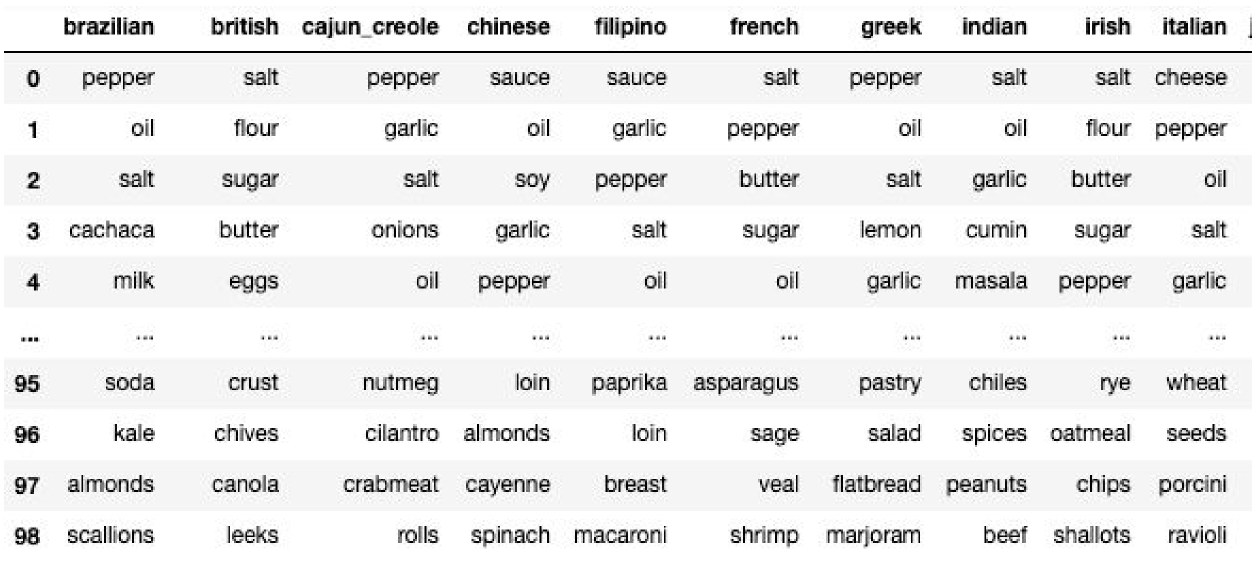
Handling NaNs in Business Dataframe

No NaNs or missing values were found in Yummly dataset

Preprocessing

I consolidated all ingredients list per cuisine type in preparation for tf-idf vectorizer. After consolidating all ingredients' text value per cuisine, I applied a TF-IDF vectorizer. TF-IDF vectorizer is used for identifying words or specific ingredients that have more weight at specific cuisine type while not appearing on other cuisines. I created two TF-IDF data frames; one based on Yummly's given ingredients list and another with n_grams (bigrams, trigrams, etc.). After applying TF-IDF, I populated the top 10 ingredients per cuisines for both TF-IDF models.

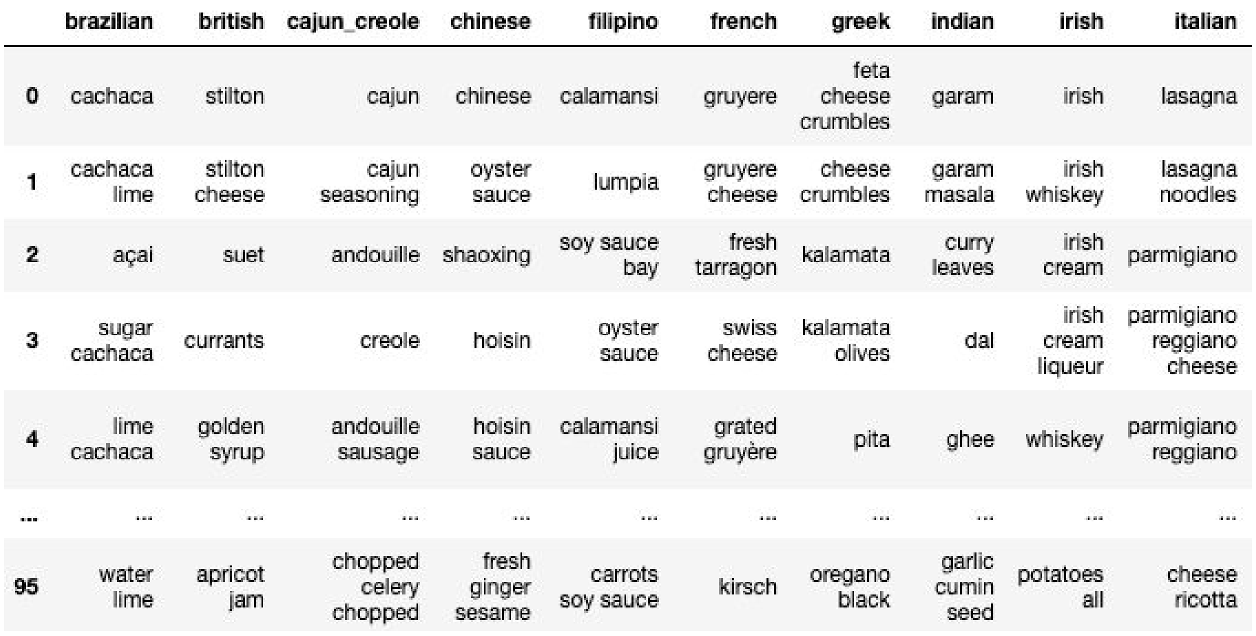
Vocab based TF-IDF



	brazilian	british	cajun_creole	chinese	filipino	french	greek	indian	irish	italian
0	pepper	salt	pepper	sauce	sauce	salt	pepper	salt	salt	cheese
1	oil	flour	garlic	oil	garlic	pepper	oil	oil	flour	pepper
2	salt	sugar	salt	soy	pepper	butter	salt	garlic	butter	oil
3	cachaca	butter	onions	garlic	salt	sugar	lemon	cumin	sugar	salt
4	milk	eggs	oil	pepper	oil	oil	garlic	masala	pepper	garlic
...	...	...	...	...	...	...	...	...	...	...
95	soda	crust	nutmeg	loin	paprika	asparagus	pastry	chiles	rye	wheat
96	kale	chives	cilantro	almonds	loin	sage	salad	spices	oatmeal	seeds
97	almonds	canola	crabmeat	cayenne	breast	veal	flatbread	peanuts	chips	porcini
98	scallions	leeks	rolls	spinach	macaroni	shrimp	marjoram	beef	shallots	ravioli

Based on the results given by Menu TF-IDF's dataframe based on Yummly's vocabulary set - the result seemed general. For instance 'garlic', 'salt', and 'pepper' are visible on all three cuisines. It did not provide any unique ingredient per cuisine as I had hoped.

N_gram based TF-IDF



	brazilian	british	cajun_creole	chinese	filipino	french	greek	indian	irish	italian
0	cachaca	stilton	cajun	chinese	calamansi	gruyere	feta cheese crumbles	garam	irish	lasagna
1	cachaca lime	stilton cheese	cajun seasoning	oyster sauce	lumpia	gruyere cheese	cheese crumbles	garam masala	irish whiskey	lasagna noodles
2	açai	suet	andouille	shaoxing	soy sauce bay	fresh tarragon	kalamata	curry leaves	irish cream	parmigiano
3	sugar cachaca	currants	creole	hoisin	oyster sauce	swiss cheese	kalamata olives	dal	irish cream liqueur	parmigiano reggiano cheese
4	lime cachaca	golden syrup	andouille sausage	hoisin sauce	calamansi juice	grated gruyère	pita	ghee	whiskey	parmigiano reggiano
...	...	...	...	...	...	...	...	...	...	...
95	water lime	apricot jam	chopped celery chopped	fresh ginger sesame	carrots soy sauce	kirsch	oregano black	garlic cumin seed	potatoes all	cheese ricotta

The N_gram based TF-IDF model provided a much better result in identifying specific words that are unique to specific cuisine compared to vocab based TF-IDF models.

Summary of Dataset

- Initial Yummly's dataset contains 20 cuisines with 39,774 rows with food ingredients.
- Consolidated Yummly's dataset based on cuisine type which reduced row count to 20 (cuisine count total)
- Applied TF-IDF Transformer onto consolidated Yummly's dataset in which n_grams (unigrams to trigrams) provided better distinction between cuisines compared to given Yummly's menu titles.

2.2 Allmenus Dataset

I decided to use *BeautfiulSoup* to webscrape allmenus.com as it contains restaurant name, cuisine type, menu item, and its menu description. All the data needed to help supplement our current datasets in recommending restaurants to the users based on their search input.

- Remove businesses from Canada and lowercase all city and state names.
- Collect restaurants menu from allmenus.com using business dataframe's states.
- Based on given restaurants' names from allmenus, filter both allmenus and yelp's business dataset.

Extracting menu data through web scraping

Prior to web-scraping, I gathered 22 states and its cities from Yelp's business dataset to be used in getting all restaurant names and menus in allmenus.com. Using restaurant information, I web-scraped and menu data frame consisting of restaurant name, menu titles, menu description and categories (cuisine).

- Gathered 1,354,448 restaurant menus from allemenus.com

Preprocessing

After gathering restaurant menus, I filtered both Yelp's business and Allmenus' menu dataset by restaurant name, which populated restaurants that appeared both in Yelp and Allmenus dataframe. It resulted in the following:

- Removed 150,439 restaurant data from Yelp's business dataset (97% reduction).
- Removed 1,040,494 restaurants from allmenus data (76.8% reduction).
- Resulted in having 1199 unique restaurants.

After filtering the dataset, I text preprocessed text values by expanding contractions, tokenizing, lemmatizing, lowercasing, and removing stop words.

Summary of Allmenus' Menu Dataset

- Before extracting data through web scraping, I retrieved US states and its respective cities that appear on Yelp's dataset.
- Used BeautifulSoup in extracting restaurant's menu data from allmenus using Yelp's states and cities information.
- Filtered restaurants that appeared on both Yelp's business and Allmenus' menu dataset which resulted in collecting 1199 unique restaurants.
- Restaurants that appeared on both allmenus and Yelp's business dataset resulted in the following:
    - Removed 150,439 restaurant data from Yelp's business dataset (97% reduction).
    - Removed 1,040,494 restaurants from allmenus data (76.8% reduction).
    - Resulted in having 1199 unique restaurants.
    - Menu dataframe contains 313,953 rows with 118 columns consisting of restaurant name, attributes, clean menu description and titles.

- ○ Created categories columns based on Allmenus' categories values (ex: There are Japanese, Indian columns, and etc.)

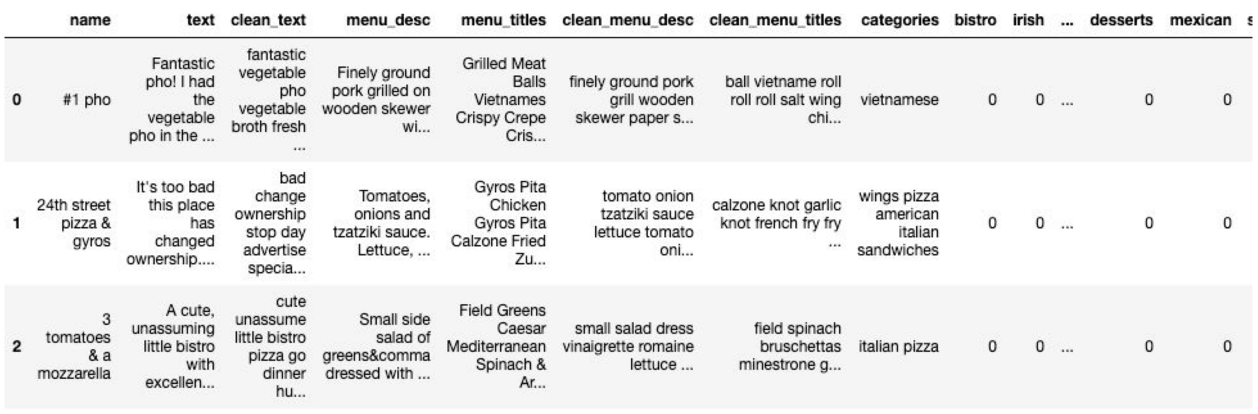
2.3 Yelp's Review and Tip Dataset

Yelp's review and tip has rich information about restaurants' attributes and users' feelings towards the restaurants they dined in. Unlike Allmenus' dataset - it does not have direct food information but rather general text value about the restaurant. Therefore, I thought it'll be useful in identifying the restaurant using text value provided in reviews and tips that illustrates different contexts other than food and drinks. Most of the columns will be removed from the business dataset as new columns will be created using Allmenus' categories columns as it has valuable information regarding restaurant's cuisine. Reviews will be filtered based on restaurants available in the filtered business dataset.

Preprocessing

- Remove all columns except business_id and restaurant name.
- Filter review and dataset by business id.
- Text preprocessing - tokenizing, lemmatizing, lowercasing, and removing stop words.
- Group all reviews/tips by restaurant name and remove duplicates. Combine text (reviews/tips), business, and allmenus dataframes to be used for EDA and to identify cosine similarities between all restaurants.

Finalized Capstone Two DataFrame



	name	text	clean_text	menu_desc	menu_titles	clean_menu_desc	clean_menu_titles	categories	bistro	irish	...	desserts	mexican
0	#1 pho	Fantastic pho! I had the vegetable pho in the ...	fantastic vegetable pho vegetable broth fresh ...	Finely ground pork grilled on wooden skewer wi...	Grilled Meat Balls Vietnames Crispy Crepe Cris...	finely ground pork grill wooden skewer paper s...	ball vietname roll roll roll salt wing chi...	vietnamese	0	0	...	0	0
1	24th street pizza & gyros	It's too bad this place has changed ownership....	bad change ownership stop day advertise specia...	Tomatoes, onions and tzatziki sauce. Lettuce, ...	Gyros Pita Chicken Gyros Pita Calzone Fried Zu...	tomato onion tzatziki sauce lettuce tomato oni...	calzone knot garlic knot french fry fry ...	wings pizza american italian sandwiches	0	0	...	0	0
2	3 tomatoes & a mozzarella	A cute, unassuming little bistro with excellen...	cute unassume little bistro pizza go dinner hu...	Small side salad of greens&comma dressed with ...	Field Greens Caesar Mediterranean Spinach & Ar...	small salad dress vinaigrette romaine lettuce ...	field spinach bruschettas minestrone g...	italian pizza	0	0	...	0	0

Summary of Yelp's Review and Tip Datasets

Most of the columns in Yelp's business, review, and tips dataframes have been removed except for business_id and text column. Grouped both review and tips dataframe and began grouping based on business_id and restaurant name which resulted in consolidating all text into one row per restaurant. After removing duplicates on all dataframes, I merged consolidated yelp's dataframe with allmenus dataframe. Merged data frame has all the necessary information in identifying the restaurant as it has menu description, menu titles, restaurant cuisine attributes, and users inputted text.

- Dropped reviews from 8,021,122 to 367,770 rows (95.5% reduction) and dropped tips from 1,320,761 to 70,834 (95% reduction)
- Combined reviews and tips dataframe which accumulated up to 438,604; however after grouping based on business_id and restaurant name - it dropped to 1199.
- Dropped rows with NaNs in the text column.
- Merged cleaned Yelp's dataframe with Allmenus dataframe on restaurant name.
- Tokenized all text and removed any whitespace occurring on text columns.
- Created 4 text dataframes:
    - All reviews dataframe: 438604 rows with 101201300 word count.
    - Positive reviews dataframe: 373687 rows with 86448479 word count.
    - Grouped positive reviews dataframe: 1198 rows with 86820968 word count.
    - Grouped reviews dataframe: 1199 rows with 101638705 word count.

Created all four dataframes to see how well it performs with doc2vec algorithm.

---

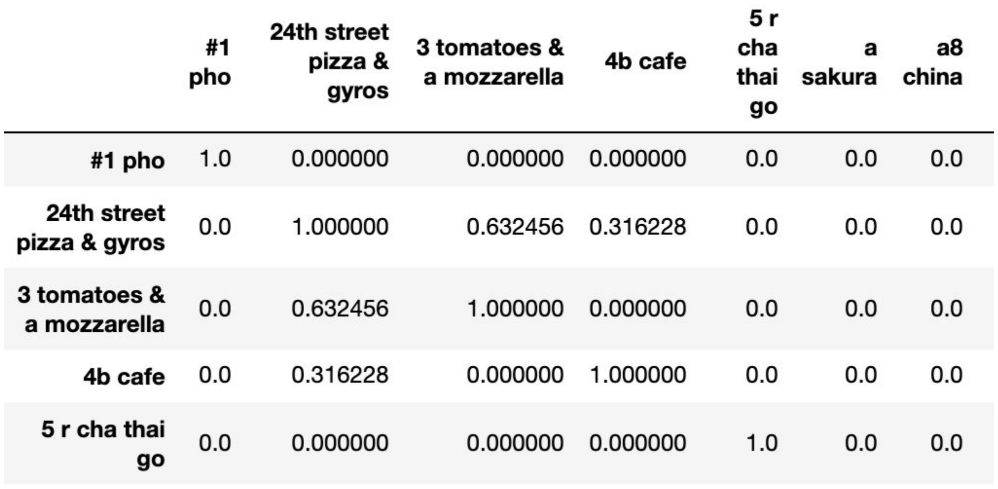
3. EDA

3.1 Visualize any similarities amongst restaurants using cosine similarity

Preprocessed dataset contains 1,199 restaurants with cuisine and attributes information (ex: Japanese, Korean, vegan, asian, gyros, dim sum, etc.). Using the aforementioned information, I wanted to see if it can accurately group restaurants together based on its given cuisine and attributes information using cosine similarity. Below are the steps I took to create a cosine similarity matrix and visualize in heatmap.

Steps:

1. Applied cosine similarity algorithm and formatted dataframe in matrix format.



	#1 pho	24th street pizza & gyros	3 tomatoes & a mozzarella	4b cafe	5 r cha thai go	a sakura	a8 china
#1 pho	1.0	0.000000	0.000000	0.000000	0.0	0.0	0.0
24th street pizza & gyros	0.0	1.000000	0.632456	0.316228	0.0	0.0	0.0
3 tomatoes & a mozzarella	0.0	0.632456	1.000000	0.000000	0.0	0.0	0.0
4b cafe	0.0	0.316228	0.000000	1.000000	0.0	0.0	0.0
5 r cha thai go	0.0	0.000000	0.000000	0.000000	1.0	0.0	0.0

2. Plotted HeatMap using matrix dataframe.

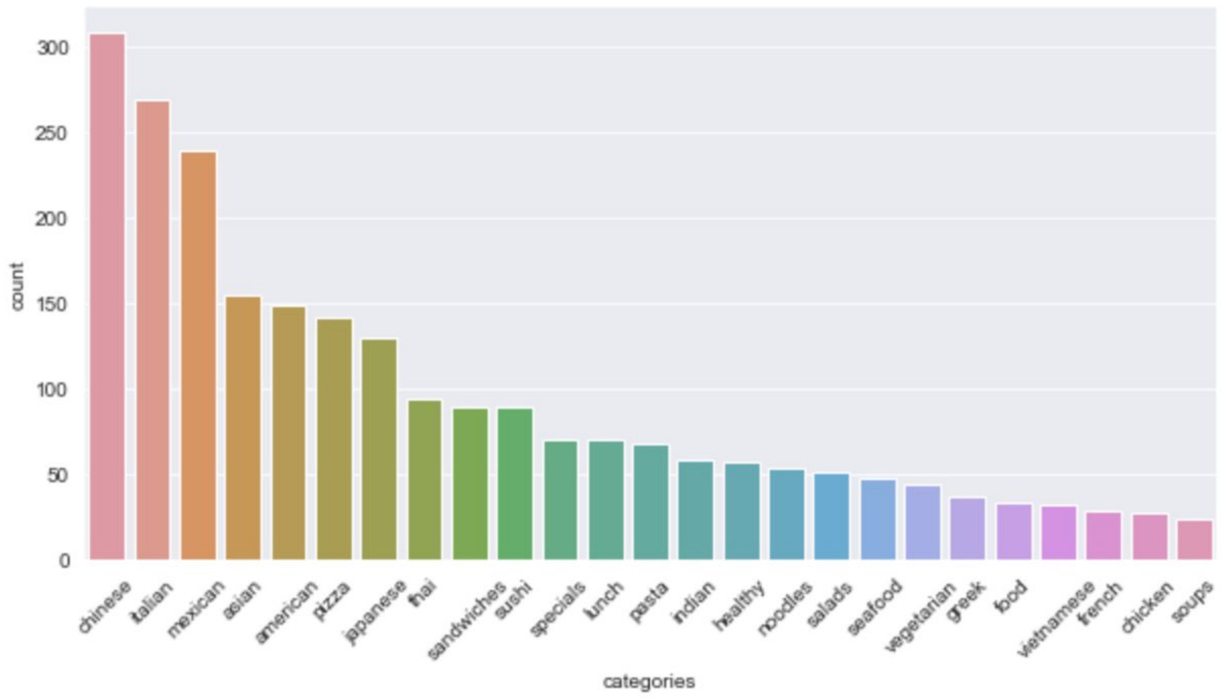
In Heatmap above, cells that get closer to red are considered to have high correlation between one restaurant to another. Considering there are red cells throughout the heatmap outside of the default diagonal line (compares restaurant to itself); it illustrates that there are many restaurants that share similar attributes and cuisine to one another.

3.2 Analyze Menu Description and Review Text

I wanted to gain data insight of the dataset with the following objectives:

- Identifying top 25 most occurring categories.
- Understand text and menu documents' word count distribution.
- Understand word count on each cuisine.
- Distribution of unigram, bigram, and trigrams.
- Visualizing top keywords of each cuisine using word clouds based on TF-IDF.

3.2.1 Identifying top 25 most occurring categories

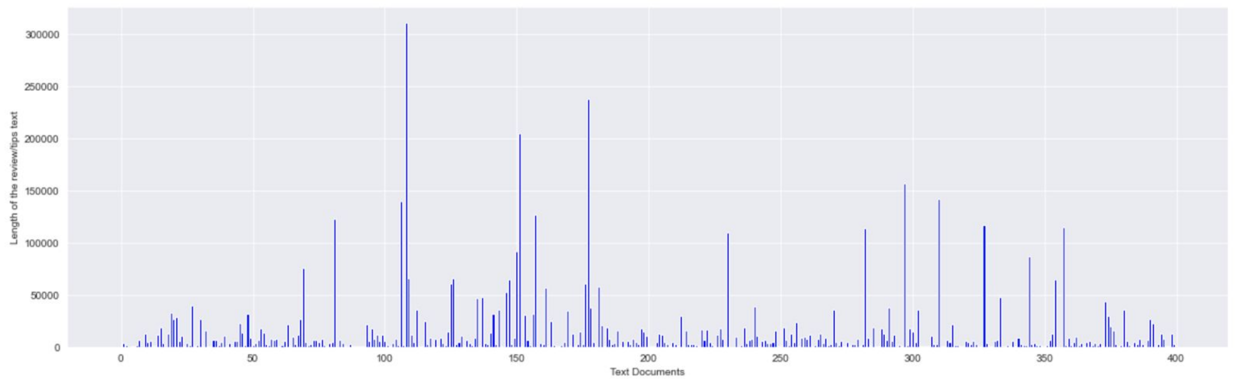


Bar graph illustrates that the dataset does not have equal counts of each cuisine; instead Chinese, Italian, and Mexican cuisines account for the majority of cuisines. However, we have to keep in mind that some restaurants have more than one cuisine type; for instance, one restaurant may be considered 'Asian Fusion' which can be Chinese, Korean, Thai, and etc.

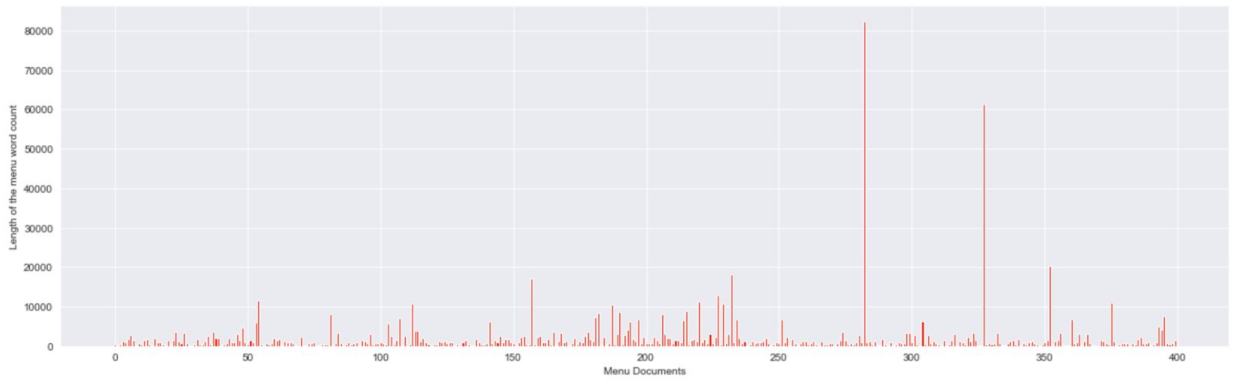
3.2.2 Understand text and menu documents' word count distribution

I wanted to gain insight on how each text and menu documents' word count is to see if the text length of each document is somewhat relatively similar to each other.

Text documents' word count



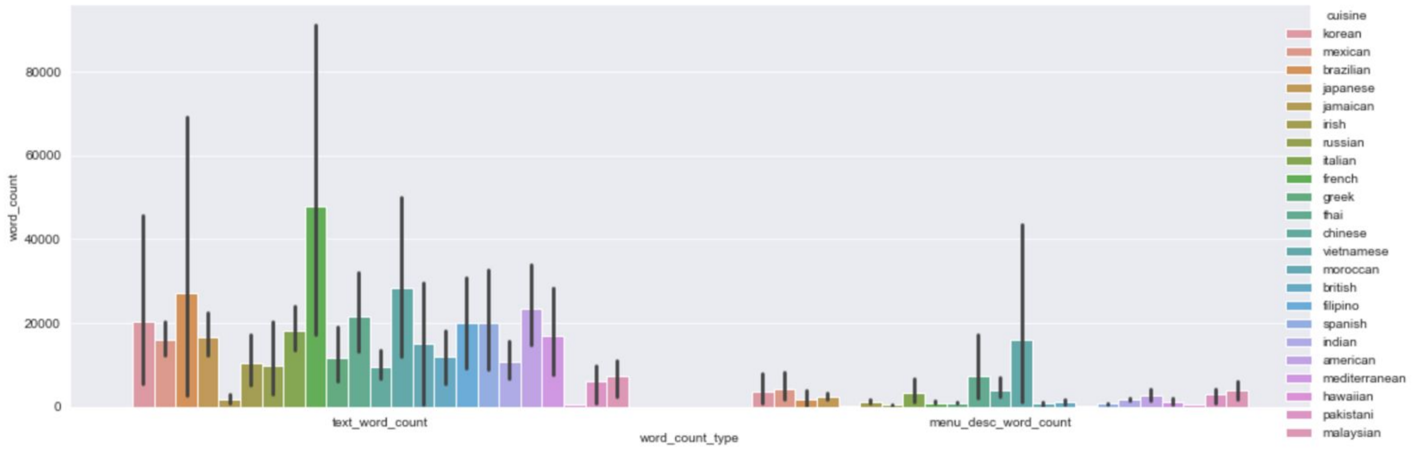
Menu documents' word count



Text column values containing users' reviews and tips have a lot word counts compared to menu column's word counts. Moreover, some documents have significantly more text information compared to other documents.

3.2.3 Understand word count on each cuisine

Having visualized both menu and text documents' word count, I wanted to understand how word count is distributed amongst cuisines.

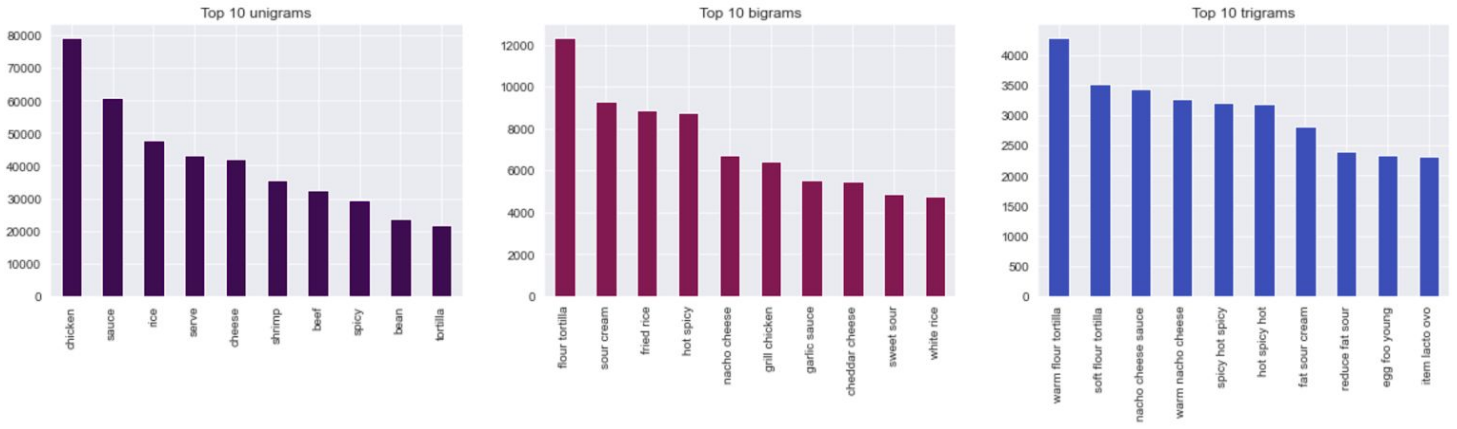


In terms of word count distribution per cuisine; French cuisine has the most word count compared to others although French cuisine restaurants appeared low in categories count bar graph. This explains that there are not that many French restaurants listed but with many reviews. Additionally, error bars are apparent on most cuisines, explaining to us that word counts are highly spread out. For instance, one document having 3,000 word count while other having less than 100 word count.

Understanding that some cuisines have higher word counts and frequencies than others; it may be a challenge in accurately identifying certain types of cuisines using doc2vec due to not having sufficient information compared to other cuisines such as French, Chinese, and Italian.

3.2.4 Distribution of unigram, bigram, and trigrams

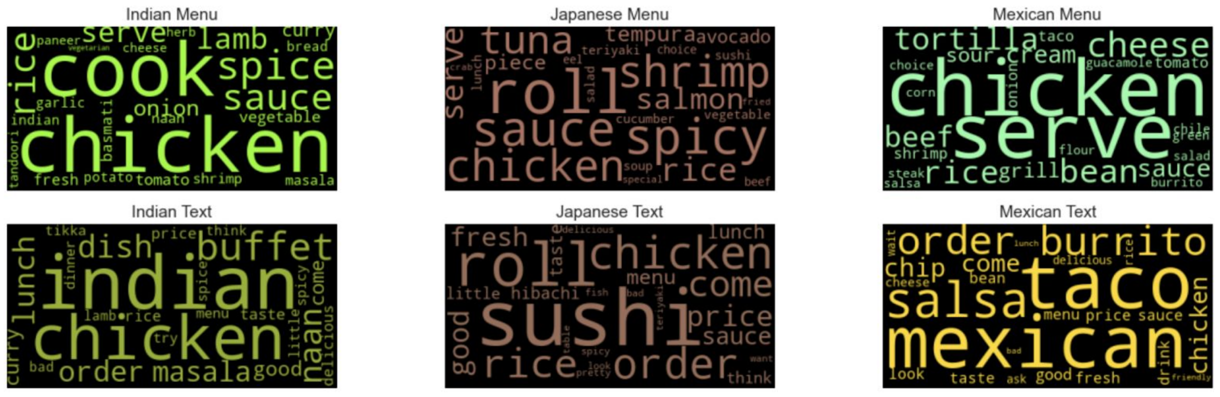
I wanted to see what are the top 10 common unigrams, bigrams, and trigrams appearing in text documents.



Based on the given information above, the top 10 most common words seem to be flour, spicy, tortilla, sauce, and milk related terms.

3.2.5 Visualizing top keywords of each cuisine using word clouds based on TF-IDF

TF-IDF measurement is used to evaluate how relevant each word is based on weight. Therefore, only retrieving import keywords for each cuisine. With the TF-IDF values, I validated whether given keywords on each cuisine based on my subjective judgement. For example, I would expect to see 'sushi' in Japanese cuisine as opposed to appearing in Irish cuisine.



Based on the word clouds above using both text and menu documents, I can conclude that the majority of the documents provided are relevant relative to its cuisine type.

3.3 Analyzing text documents through LDA modeling

Unlike menu documents, text documents consists of Yelp's review and tips dataset and text information isn't primarily always focused on food and drinks; it also contains information regarding restaurants' service, ambience, and other miscellaneous information. Therefore, I wanted to see if any hidden structure in a collection of texts uncovered through topic modeling can be useful for the search engine.

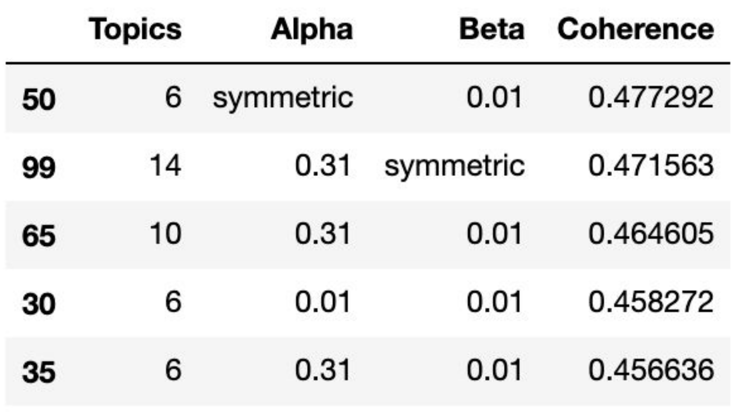
In preparation for LDA model, I used gensim library in phrase modeling (building bigrams and trigram), data transformation such as creating dictionary (unique id for each word in the document), building bag of words corpus after filtering words that appeared less than 10 time and appeared more than 50% of all documents and keeping only 100,000 most frequent tokens. Lastly, applied the TF-idf model to gain weight values per word.

3.3.1 LDA Hyper-parameters tuning

I primarily gauged LDA model performance based on coherence scores. Coherence score measures the degree of semantic similarity between high scoring words in the topic which helps distinguish between topics that are semantically interpretable topics and topics that are artifacts of statistical inference. In other words, the higher the score (0 - 1) the better it is in distinguishing one topic from another thus being unique. I built multiple LDA models using one of my helper functions in which iterates over a number of topics, alpha values, and beta values. Below are the following parameters I used in building LDA models.

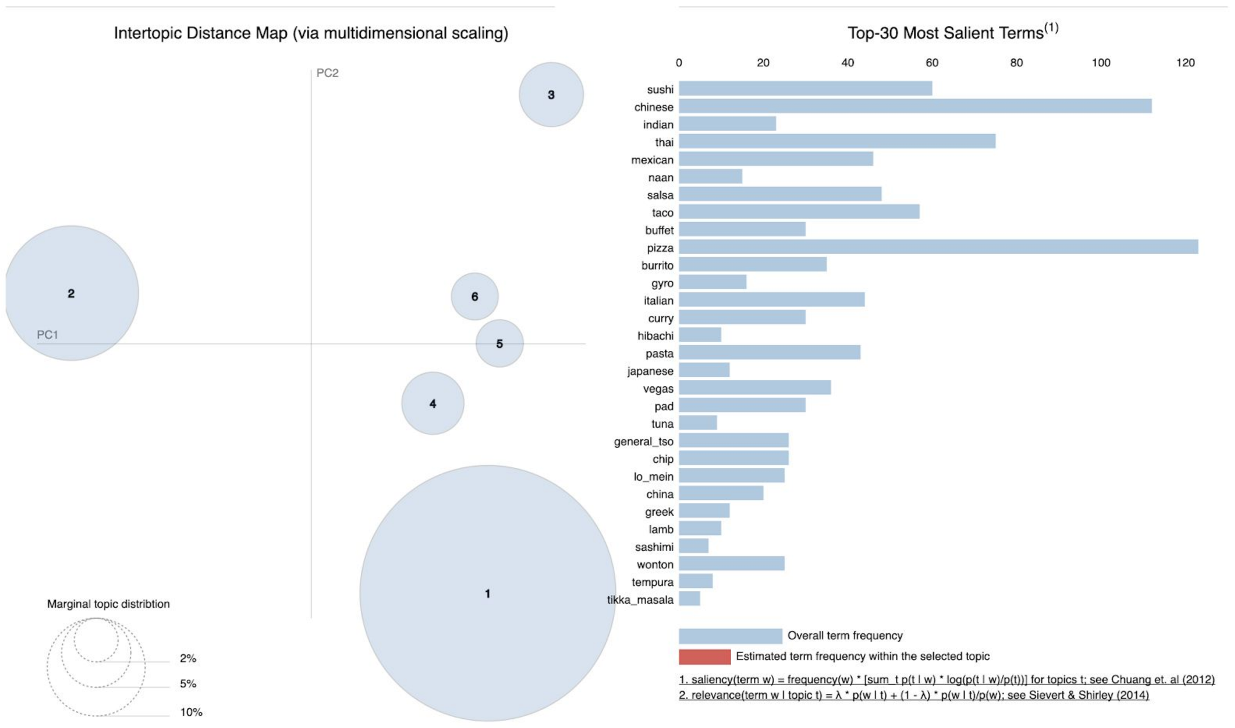
- num_topics = number of requested latent topics to be extracted from corpus
  - Started from 2 topics up to 20 topics with increments of 4
- chunksize = number of docs to be used in each chunk
- alpha = document-topic density
  - started from 0.01 up to 1 with increments of 0.3
- eta = word-topic-density
  - started from 0.01 up to 1 with increments of 0.3
- passes = number of pases through the corpus (epochs)

I tuned num_topics, and alpha, and eta (beta) values in getting the best possible LDA model based on given coherence scores of each model. I tuned alpha and beta values to produce more or less specific topic distribution per document and more or less word specific distribution per topic, to visually see how it impacts LDA model performance.



	Topics	Alpha	Beta	Coherence
50	6	symmetric	0.01	0.477292
99	14	0.31	symmetric	0.471563
65	10	0.31	0.01	0.464605
30	6	0.01	0.01	0.458272
35	6	0.31	0.01	0.456636

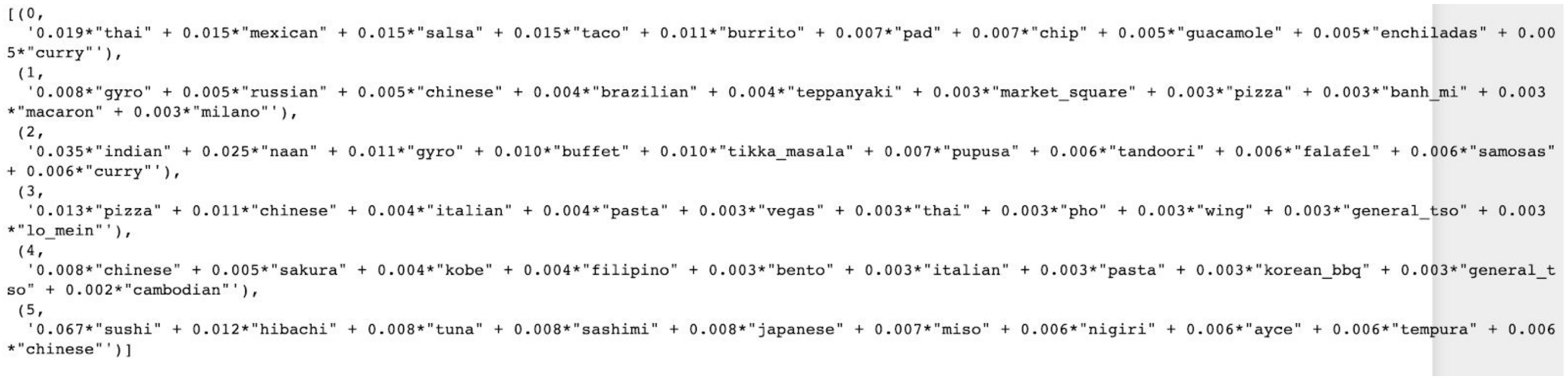
Based on the report from my helper function, LDA model with 6 topics, symmetric alpha, and beta value of 0.01 got the best performance with coherence score of 0.48



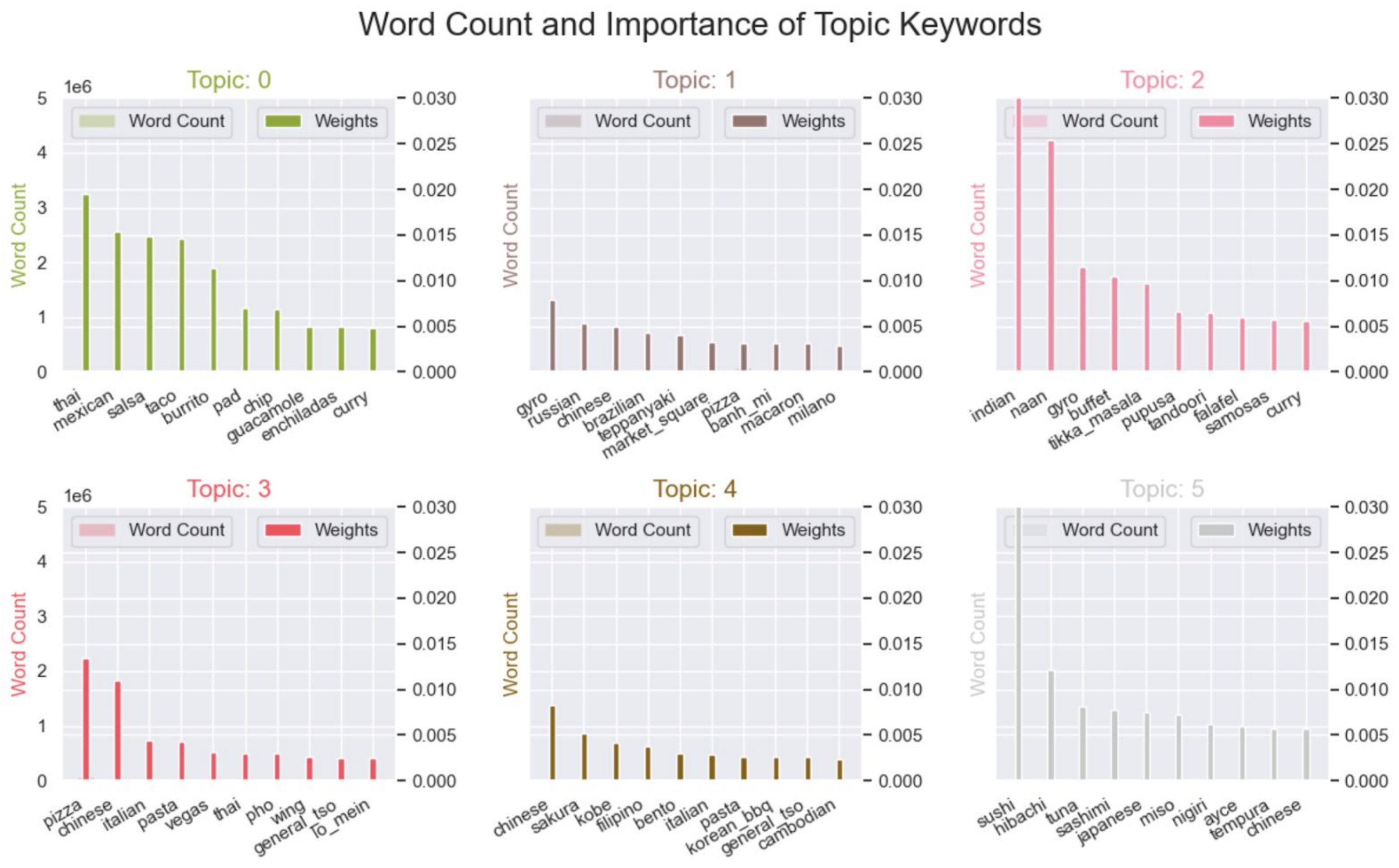
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

3.3.2 Visualizing top 10 keywords per topic

Below are the 6 topics that were populated with the given coherence result of 0.48.



```
[(0,
  '0.019*"thai" + 0.015*"mexican" + 0.015*"salsa" + 0.015*"taco" + 0.011*"burrito" + 0.007*"pad" + 0.007*"chip" + 0.005*"guacamole" + 0.005*"enchiladas" + 0.00
5*"curry"'),
 (1,
  '0.008*"gyro" + 0.005*"russian" + 0.005*"chinese" + 0.004*"brazilian" + 0.004*"teppanyaki" + 0.003*"market_square" + 0.003*"pizza" + 0.003*"banh_mi" + 0.003
*"macaron" + 0.003*"milano"'),
 (2,
  '0.035*"indian" + 0.025*"naan" + 0.011*"gyro" + 0.010*"buffet" + 0.010*"tikka_masala" + 0.007*"pupusa" + 0.006*"tandoori" + 0.006*"falafel" + 0.006*"samosas"
+ 0.006*"curry"'),
 (3,
  '0.013*"pizza" + 0.011*"chinese" + 0.004*"italian" + 0.004*"pasta" + 0.003*"vegas" + 0.003*"thai" + 0.003*"pho" + 0.003*"wing" + 0.003*"general_tso" + 0.003
*"lo_mein"'),
 (4,
  '0.008*"chinese" + 0.005*"sakura" + 0.004*"kobe" + 0.004*"filipino" + 0.003*"bento" + 0.003*"italian" + 0.003*"pasta" + 0.003*"korean_bbq" + 0.003*"general_t
so" + 0.002*"cambodian"'),
 (5,
  '0.067*"sushi" + 0.012*"hibachi" + 0.008*"tuna" + 0.008*"sashimi" + 0.008*"japanese" + 0.007*"miso" + 0.006*"nigiri" + 0.006*"ayce" + 0.006*"tempura" + 0.006
*"chinese"')]
```



According to the words given per topic; here are the following topic titles I can assume:

- Topic 0 - Hispanic/Mexican
- Topic 1 - Traditional dishes or places(?)
- Topic 2 - Indian
- Topic 3 - Bread, Noodles, and starch (?)
- Topic 4 - Meat(?)
- Topic 5 - Japanese

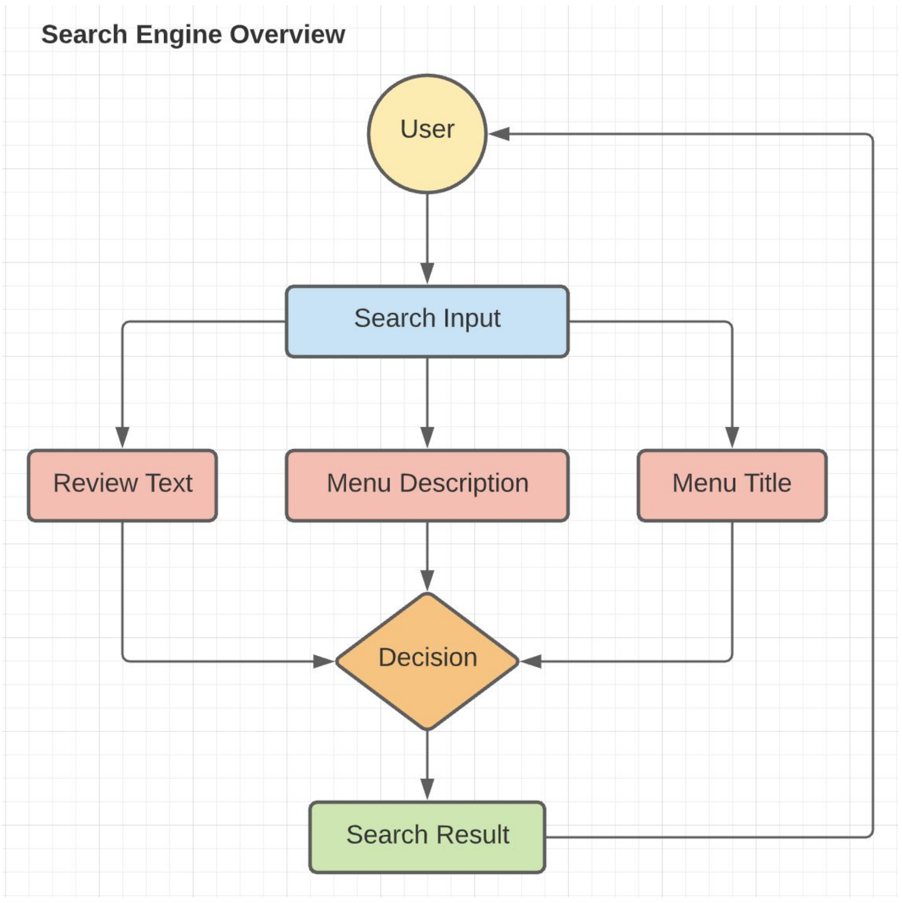
Topic 1, 3, and 4 continues to be a bit vague on what they're topic is supposed to be. However I can speculate that Topic 3 could be related noodle dishes since 'pasta', 'pho', and 'lo mein' appeared as having contributions to the topic. Topic 4 could be about meat related diets since 'kobe', 'korean_bbq' and 'general_tso' came out to be top 10. Topic 1 remains to be a mystery, as it seemed to be mixed with multiple topics, if I were to make a guess, I would assume it's related to traditional dishes or places.

4. Machine learning

In this machine learning file, I'll be creating three Doc2Vec models to represent words and documents into numerical vectors. Each Doc2Vec model will be built and trained using Yelp's review/tips, menu description, and menu titles. The idea behind building three Doc2Vec models is to consolidate search results which consists of most similar documents based on the user's search query using cosine similarity and built-in Doc2Vec's most_similar method. The top 10 documents will be chosen in one of following ways:

1. When there are common documents populated in all three results; the document will be added as part of the final result. For instance, when a restaurant named 'Panda Express' is shown in all three Doc2Vec results, Panda Express will be shown as part of the final result visible to the user.
    a. When similar restaurant names do not accumulate up to 10; the rest will be determined based on cosine similarity score.
2. When no common documents are found through three Doc2Vec results, all three results will be consolidated and results will be shown based on cosine similarity score in descending order.

Below is overview of NLP search engine's control flow:



4.1 Building Doc2Vec Models

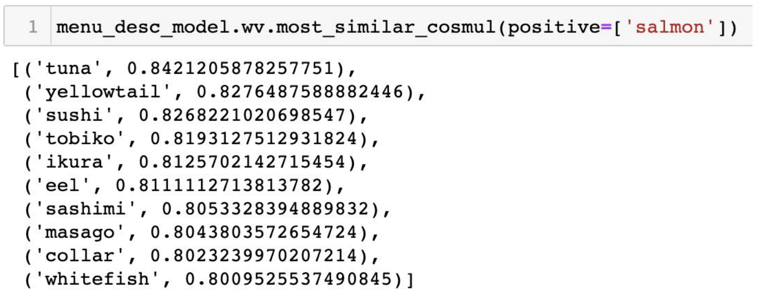
I created a distributed memory model with a size of 100 (100 vectors). With window size of 2 for menu dataframe and 10 for text dataframe. Smaller window size is inputted for menu dataframe as it does not have extensive text count compared to text dataframe. Set dbow_words parameter to 1, for all Doc2Vec models which train word vectors in skip-gram fashion. Sample parameters are also inputted to remove high frequency words like 'the' during model building. Negative sampling is applied to computationally speed up the process when finding similar vectors through at random instead of looking through all other words.

4.2 Evaluating words embedding

I evaluated models by checking top 10 similar words based on given input to ensure if the result made sense or not. When the model returned in accurate results, I created vector column for each dataframes which will be used during search method for retrieving most similar documents based on user search input.

Below are the results from three Doc2Vec models that contains its own vocabulary:

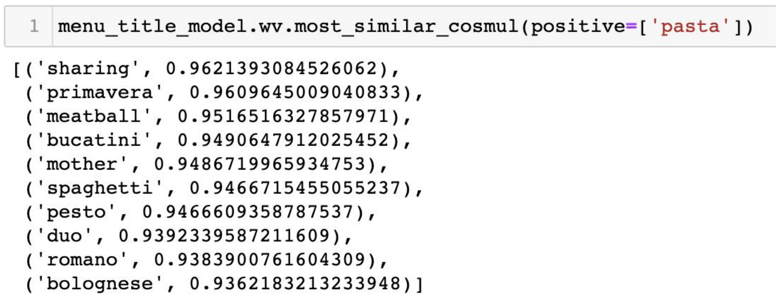
- Menu Description's Doc2Vec model



```
menu_desc_model.wv.most_similar_cosmul(positive=['salmon'])
```

```
[('tuna', 0.8421205878257751),
 ('yellowtail', 0.8276487588882446),
 ('sushi', 0.8268221020698547),
 ('tobiko', 0.8193127512931824),
 ('ikura', 0.8125702142715454),
 ('eel', 0.8111112713813782),
 ('sashimi', 0.8053328394889832),
 ('masago', 0.8043803572654724),
 ('collar', 0.8023239970207214),
 ('whitefish', 0.8009525537490845)]
```

- Menu Title's Doc2Vec model



```
menu_title_model.wv.most_similar_cosmul(positive=['pasta'])
```

```
[('sharing', 0.9621393084526062),
 ('primavera', 0.9609645009040833),
 ('meatball', 0.9516516327857971),
 ('bucatini', 0.9490647912025452),
 ('mother', 0.9486719965934753),
 ('spaghetti', 0.9466715455055237),
 ('pesto', 0.9466609358787537),
 ('duo', 0.9392339587211609),
 ('romano', 0.9383900761604309),
 ('bolognese', 0.9362183213233948)]
```

- Review text's Doc2Vec model

```
test = text_model.wv.most_similar_cosmul(positive=['pho'])
test
```

```
[('vietnamese', 0.8965592384338379),
 ('vermicelli', 0.8624076843261719),
 ('biet', 0.8565489053726196),
 ('viet', 0.8530400395393372),
 ('saigon', 0.8467001914978027),
 ('dac', 0.8427303433418274),
 ('banh', 0.8308336734771729),
 ('phos', 0.8118297457695007),
 ('bahn', 0.8098064064979553),
 ('thanh', 0.8051256537437439)]
```

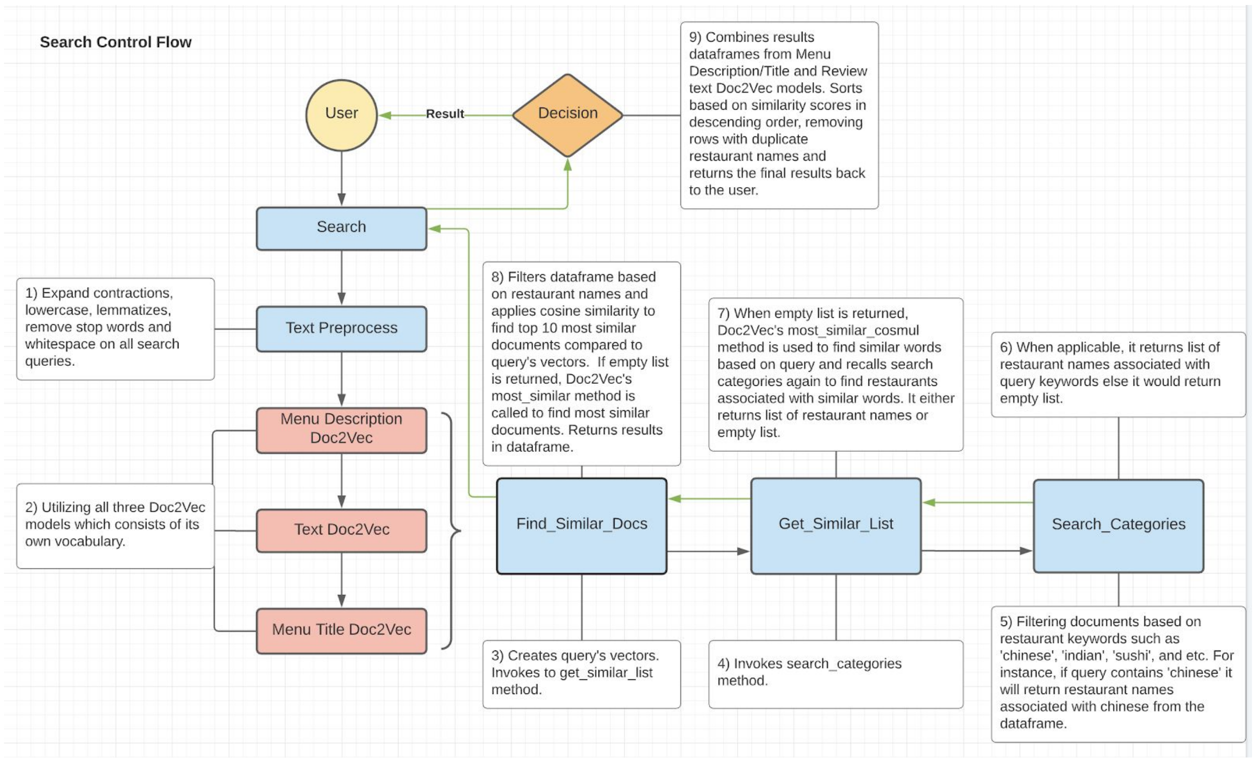
Based on the results above, all the inputs added to all three Doc2Vec models produced coherent results. For instance, most similar words to 'pasta' were meatball, pesto, spaghetti, and etc. while 'salmon' returned sushi, tuna, eel, sashimi, and etc. Therefore, I was able to conclude that models are built correctly with appropriate hyperparameters.

4.3 Search Control Flow

Search method includes the following methods:

- **find_similar_docs method**: Finds similar documents based on vectors.
- **get_similar_list method**: Finds and gathers documents that have specific keywords associated with query.
- **search_categories method:** Searches for restaurants that share similar restaurant keywords such as 'Chinese', 'Indian', 'sushi', and etc.
- **expand_contractions method**: Expands contractions.
- **Tokenize method**: Text preprocessing.

Below is the chart how results are returned based on query:



9) Combines results dataframes from Menu Description/Title and Review text Doc2Vec models. Sorts based on similarity scores in descending order, removing rows with duplicate restaurant names and returns the final results back to the user.

1) Expand contractions, lowercase, lemmatizes, remove stop words and whitespace on all search queries.

2) Utilizing all three Doc2Vec models which consists of its own vocabulary.

8) Filters dataframe based on restaurant names and applies cosine similarity to find top 10 most similar documents compared to query's vectors. If empty list is returned, Doc2Vec's most_similar method is called to find most similar documents. Returns results in dataframe.

7) When empty list is returned, Doc2Vec's most_similar_cosmul method is used to find similar words based on query and recalls search categories again to find restaurants associated with similar words. It either returns list of restaurant names or empty list.

6) When applicable, it returns list of restaurant names associated with query keywords else it would return empty list.

3) Creates query's vectors. Invokes to get_similar_list method.

4) Invokes search_categories method.

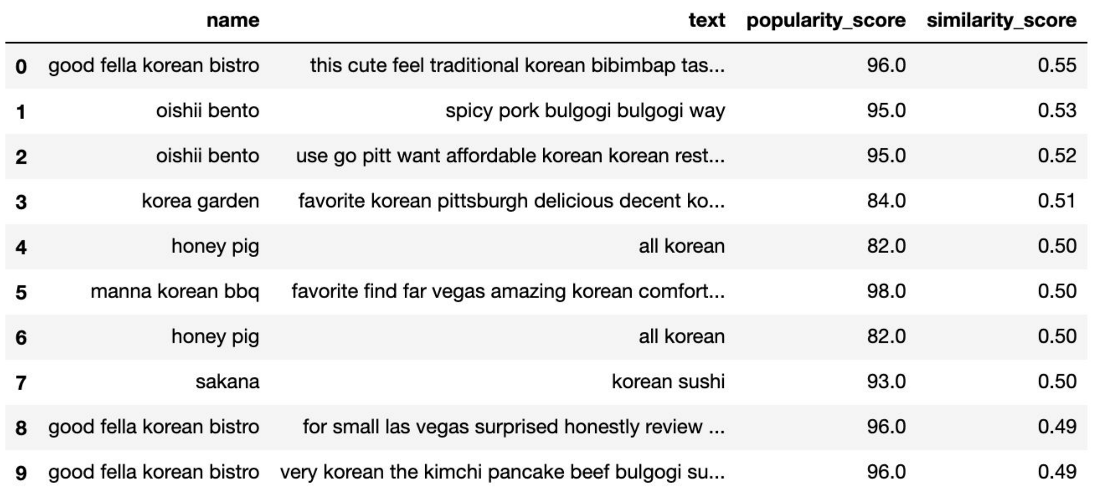
5) Filtering documents based on restaurant keywords such as 'chinese', 'indian', 'sushi', and etc. For instance, if query contains 'chinese' it will return restaurant names associated with chinese from the dataframe.

1. Expand contractions, lowercase, lemmatizes, remove stop words and whitespace on all search queries.
2. Utilizes all three Doc2Vec models which consists of its own vocabulary
3. Creates query's vectors and invokes get_similar_list method.
4. Invokes search_categories method.
5. Filtering documents based on restaurant keywords such as 'Chinese', 'Indian', 'sushi', and etc. For instance, if a query contains 'Chinese' it will return restaurant names associated with the word 'Chinese' from the dataframe.
6. When applicable, it returns a list of restaurant names associated with query keywords else it would return an empty list.
7. When an empty list is returned, Doc2Vec's most_similar_cosmul method is used to find similar words based on query and recall search categories again to find restaurants associated with similar words. It either returns a list of restaurant names or an empty list.
8. Filters dataframe based on restaurant names and applies cosine similarity to find top 10 most similar documents compared to query's vectors. If an empty list is returned, Doc2Vec's most_similar method is called to find most similar documents.
9. Combines results dataframes from menu description/title and review text Doc2Vec models. Sorts based on similarity scores in descending order after removing rows with duplicate restaurant names and returns the final results back to the user.

4.4 TF-IDF Search Engine

Three trained Doc2Vec models provided satisfactory results in returning most similar documents based on given user's search input. I also wanted to check whether the TF-IDF model would provide a good result by creating a similarity matrix between a list of keywords.

After creating the TF-IDF model, I inputted the search as 'korean bulgogi food' and returned the following result using Gensim's MatrixSimilarity method.



	name	text	popularity_score	similarity_score
0	good fella korean bistro	this cute feel traditional korean bibimbap tas...	96.0	0.55
1	oishii bento	spicy pork bulgogi bulgogi way	95.0	0.53
2	oishii bento	use go pitt want affordable korean korean rest...	95.0	0.52
3	korea garden	favorite korean pittsburgh delicious decent ko...	84.0	0.51
4	honey pig	all korean	82.0	0.50
5	manna korean bbq	favorite find far vegas amazing korean comfort...	98.0	0.50
6	honey pig	all korean	82.0	0.50
7	sakana	korean sushi	93.0	0.50
8	good fella korean bistro	for small las vegas surprised honestly review ...	96.0	0.49
9	good fella korean bistro	very korean the kimchi pancake beef bulgogi su...	96.0	0.49

With the given result, I can safely conclude that the TF-IDF search model also returns coherent results. As search input representing korean dishes returned restaurants that are associated with korean dishes.

4.5 Summary

I built two search engines using Doc2Vec and TF-IDF. Three Doc2Vec models are created and trained using menu text values and Yelp's review text values. Unlike the TF-IDF approach, Doc2Vec models are achieved in identifying semantic proximity between words/documents and finding the meaning of the words. Although the TF-IDF search engine provided coherent results, it only looks for words that are listed in keywords that match the document, in other words, it does not perform well when never seen words are inputted. Overall, Doc2Vec is a useful algorithm in finding meaning of the documents, however results may vary based on hyper-parameters which require some tinkering.

5. Suggested Improvement

There are several ways to improve performances in identifying similar restaurants based on the user's search query.

- Using all of Yelp's review text data instead of trimming down based on what was available on allmenus.com. At the end of my NLP project, I recognized that using a combination of LDA and Word2Vec/Doc2Vec, I could have found restaurants' cuisine type without use of allmenus' menu text data through topic modeling. Using this approach, may have provided more restaurants to search.
- Currently, the search engine populates based on similarity score only, the search result may be more relevant when populating list based on user's location. For instance, if a user queried from Los Angeles, it would show restaurants near that area.
- Utilizing Spark or Dask in speeding up computational time in text pre-processing and populating search results.

6. Project Summary

- Search engine was built with the purpose of finding most similar restaurants based on user query whether it may be food, drink, or any restaurant's attributes.
- During the EDA process, we were able to differentiate and visualize text values based on cuisines (ex: 'Japanese', 'Chinese', 'French', and etc.).
- Created three Doc2Vec models based on Yelp's review text data and allmenus' menu data which provided coherent results in providing good results.
- The TF-IDF search model was created based on keywords which also provided coherent results but unlike Doc2Vec models it does not understand meaning of the words and semantic relationships between words and documents.

Having restaurants’ text information from Yelp’s reviews/tips and its restaurants’ menu data provided sufficient text information in understanding the semantic relationship between documents using the Doc2Vec algorithm. Having three Doc2Vec models to compare the result and gathering highest cosine similarity from the user's query turned out to be effective in providing coherent results back to the user.