Improving Yelp Recommendation System

SFU Programming for Big Data Lab I. Fall 2018
Denise CHEN (301375388)
Mark Hoi Victor HAU (301376384)
Yunhye JOO (301378917)

Defining the Problem

1. Limited Search Fields on Yelp

On the Yelp app or website, there is only two fields - types of cuisine and location. However, we often want to search based on multiple attributes such as an economical Chinese restaurant that is ideal for friends gathering. It is difficult to search with other characteristics of restaurants. We want to build a model that extract the most important keywords from reviews that can help analyze the quality of the restaurants as well as how relevant the restaurants are with the input query. We can process the reviews and calculate how similar the review vector with the query vector is. We can then rank and recommend the restaurants to the user.

2. Deception of High Rating

Sometimes we would feel that certain restaurants are overrated if we blindly follow the number of stars the restaurants have to make our choice for the dining option. This may happen because different people have different tastes and standards for restaurants. We often feel disappointed when there is a mismatch between our taste and the taste of the review writers. At times, we would even doubt how real the reviews are and whether they are fake reviews written by people who get paid for doing that. Does Yelp ask for money from restaurants so that the negative reviews disappear? Yelp not only allows customers to rate a restaurant on a scale of one to five stars, but also encourages customers to write reviews about their dining experience. Five-star reviews may be written with different interpretation of what these stars represent. Based on the review written by the users, the text-rich reviews are more indicative of restaurants' properties and user preferences. However, reading all reviews one by one is tedious and time-consuming. We plan to build a model to produce sentiment score of each review and the recommendation can suggest the restaurants with high sentiment score, so users can visit the restaurant not only depending on the star rating.

3. Lack of Features to Predict Users' Preferences Based on Users' Data

On Yelp, people mainly just input keywords and search for the restaurants. Some certain users may have routine of visiting particular restaurants at a particular time or have preferences towards restaurants sharing similar properties and quality. Thus, we can take the restaurants visited by users into account and recommend suitable restaurants that they desire to dine at. Based on the restaurants previously visited by Yelp users, we are eager to suggest same types of restaurants that match with the restaurants visited by users with item-item similarity, search for restaurants that other users with similar preferences visited with user-item similarity, and find the top-rating restaurants visited by users' friends. We indicate the algorithm each suggestion comes from when we output the results, such that Yelp users can get restaurants recommendations based on larger range of factors.

Methodology

- **Data Preprocessing and Analysis:** We want to focus on the city that has the largest review count. Las Vegas has the largest number of reviews, so we filter out the reviews in Las Vegas. We filter all stores that are not in categories as food or restaurants. Then, we store the filtered data into database and use this dataset for item-item similarity method of recommendation system. Regarding business data, each store has several categories and they are stored as a list. We split the categories and convert categories into a tuple with business IDs. Then, we analyze the top categories among all stores from the Yelp dataset. Attributes feature in business data are stored as a dictionary. We select the business ID and attributes and read them into RDD as tuple. Then, we use python regular expression to split the attributes stored in dictionary, and analyze which are top attributes from all stores in Yelp dataset. Feature hours are also stored as dictionary type. We use the same method as above and write other regular expression to extract the key value pair, and see what time the store opens in a week. Based on the hours feature, we also analyze what are the most popular hours that the store opens. For check-in data, the count of check-ins within each hour is also a dictionary. We analyze the most frequent check-in hours for each business, and also group by the category of stores to analyze the popular check-in hours for each category of business.

- **Pyspark Pipeline:** We use the review data for NLP analysis and generate the frequent topics from each review of business. We concatenate reviews of each business and map each review and business ID as tuple. Then, we use Pyspark **RegexTokenizer** to tokenize the review, and Pyspark **StopwordsRemover** to filter out the stop words

which are not needed for NLP analysis. Afterwards, we use TF*IDF method to vectorize the review and use Pyspark **CountVectorizer** and **IDF** methods. We feed all the above methods into pipeline and transform the reviews through the pipeline.

- **Feature/Topic modeling:** For feature modeling of review data, we use **Word2Vec** and **LDA** methods, and compare the similarity score of these two methods. Word2Vec translates each word into a vector of 100 features. These vectors are located in a feature space of 100 dimensions, with similar words closer together and unrelated words further apart. LDA is a method to classify the words into certain topics. LDA would output the matrices of the word with each topic and compute the probability of the connection between each topic and every word of review. Once the words are translated into vectors, we perform **cosine similarity** formula to determine how similar two words are based on cosine similarity. Mathematically, cosine similarity is the dot product of the two vectors divided by the product of the magnitudes of those two vectors. This calculates the angle between two vectors. The result of LDA method outputs the higher similarity score than the Word2Vec method. For instance, we input keyword as chicken cheese burger, these two table are the results of LDA and Word2Vec respectively.

--Table of Word2Vec method

Business_ID	Similarity Score	Categories
EUWBT5GDxPC95w9itZ1EHw	0.6028132	Buffets, Restaurants
M4vh_kzppP1nsxo7hsaVIA	0.577314	Restaurants, Buffets
5iHctUjkQTGwEvOaBkwMRQ	0.5727653	Breakfast & Brunch, American (Traditional), Food, Fast Food, Restaurants

--Table of LDA method

Business_ID	Similarity Score	Categories
xnVkYE3iMp_aZniiCIuD0g	0.9978023	Breakfast & Brunch, Food, Restaurants, American (Traditional)
N1AujkudU6TD4LqpFg72SQ	0.99546835	Specialty Food, Food
xowy4YxBXXGOYuajsxjwkA	0.99325532	Food Trucks, Caterers, Restaurants

- **Keras Neural Network:** Our goal is to create a neural network to predict the sentiment of review as positive or negative based on the language used in review data. Restaurant reviews with rating of 3 or 4 stars were thrown out due to the lack of consistency between reviewers. The low rated reviews were given a rank of zero and the high rated ones were given a rank of one. A neural network was set up with keras package using a convolutional filter, pooling filter, and both ReLU and sigmoid activation function to predict the sentiment of each review as either 0 or 1. The final model was 96% accurate on prediction score.

- **Pyspark ml:** For collaborative filtering, we use user-item similarity method to recommend which restaurants the user may like. We use Pyspark ALS library to produce matrix factorization of the user ID and restaurant ID. For new user that has no rating history, we use drop method to drop any rows in the DataFrame of predictions that contain NaN values. At first, we get the rmse score as 2.5. After the hyperparameter tuning and cross-validation, we are able to reduce rmse to 1.5 which output as a better performance.

- **Visualization:** We use Tableau to output the plot of business and user data. We look into the rating distribution of every business. We also analyze the review count of each state in U.S.A. We find out that NV and AZ state have the most reviews, so we look into cities in these 2 states. Then, we find out city of Las Vegas has the most review. For the top 10 review count of cities, we generate the chart plot to see the category of distribution of each review. Besides, we look into stores' locations of store distribution in Las Vegas city. Each circle size depends on the total review count, so the larger the circle, the more review it has for each store. In addition, we use Python wordcloud package to see the most frequent words appeared in positive and negative reviews. We also generate friends social network plot to see top 5 review count users. Each line of the top 5 user is connected to their friends' userid. At last, we output the donut chart of each category of store and see the percentage of star rating for each category.

  Link of Tableau:
  https://public.tableau.com/views/YelpDatasetVisualization/Dashboard1?:embed=y&:display_count=yes

- **Flask:** We use Flask templates to generate and render web pages that can deliver requests and receive data from Spark tasks. The forms on the webpages are generated with the help of WTForms library such that the values are obtained once a POST is submitted. The html files with basic layout are stored in templates folder. The missing parts dependent on real-time data are substituted by Flask by routing the web address to functions that place the data.

- **Celery and RabbitMQ:** In order to run the PySpark codes on the university Big Data Lab cluster or locally, we need to connect our Flask templates with Celery workers by using a broker, which is RabbitMQ in our choice. Using Celery can prevent the web browser from being frozen as the workers work behind at the background to tackle the issue of taking a long time to run the processes. Spark codes cannot directly communicate with Flask nor Celery. Thus, we need to use the Celery-Spark library written by our professor Prof. Gregory Baker to set up a Spark session to be used in the cloud cluster or local machine. The Spark tasks can thus run like Celery tasks at the background communicating with Flask via RabbitMQ.

- **Process in our project:** Depending on whether it is a GET or POST requests of the webform, the Flask app either generates the default map we set or send the spark tasks to the Celery-Spark worker via RabbitMQ. After the Spark ETL and machine learning computation is done, the resulted Spark Dataframe Row object is then received by the Flask app. The app then generates pins with infobox on the Google Maps based on the data attributes such as latitude, longitude, ranking and score that Flask receives from the worker.

- **Google Maps API:** We use Google Maps API to embed the real interactable Google Maps with satellite images and even Pegman Street View as available functions. We set up a Google Cloud account, obtain a unique key and get charged every time a request to load the Google Maps is fulfilled. We integrate the Google Maps into the Flask template by building on the Flask-GoogleMaps library. We use three different colors of Google Maps pins to represent three different types of results for collaborative filtering. We put red pins ranking from A to Z for the top 26 results for the content-based analysis by incrementing picture path names and red pins for the rest.

- **Port redirection:** We redirect the 127.0.0.1(localhost) port 5000 of the individual session of the cloud cluster machine to the localhost port 5000 of our local machine. The web application can thus be simply accessed on any browser by imputing localhost:5000 on the address bar.

- **Python virtual environment:** In order to install packages in SFU cloud, we set up a virtual environment to do it.

Problems

High computation while training Word2Vec and ALS model: When training the Word2Vec model to review data, we often encounter the problem of running out the memory since the package would build up neural network to train the data. We need to filter the data into the smaller scope in order to successfully output the result through Word2Vec. For ALS model, it takes 2 hour to train model which takes too long time. And, we also modify and tune the model several times in order to get the better result of rmse which takes us a huge amount of time to successfully get the better result of the model.

Different city locations of the recommendation system output for existing users: When we recommend the restaurants for existing users based on the user id, several restaurants would pop out in different cities. When we find similar restaurants based on the restaurants visited by the user before, we would often get the restaurants located in different cities. Besides, when we recommend the restaurants from the rating given by users' friends, the restaurants would often fall into different cities too. To solve this problem, we need to filter the restaurants that are not in the same city.

Running Flask app locally: We choose to run our entire codes in the SFU cluster because there is a limitation on the computing power of the local machine we have. Also, there is a conflict with version compatibility among Python dependencies, Hadoop and Spark. Such conflict cannot be solved using virtual environment despite numerous hours of effort to try to fix such problems. The problem prevents parquet from being read, and also the memory of our local machines is not big enough to hold the cache during consecutive computation of several Spark ETL and machine learning tasks.

Localhost port redirection: We faced another new issue when trying to run the Flask app with Celery-Flask app as an app within on the school cluster. At first, we did not know how to use the browsers in our local machine to view the web application although the app successfully ran on the cluster's localhost port 5000 of the individual user we log in as. We later found out we can redirect/connect the localhost of remote machine to local machine.

Integrating Flask with Celery-Spark: It is difficult to use Celery-Spark library with Flask at first when we can only learn how to run Flask with Celery and RabbitMQ from online tutorials. There is a configuration issue with Spark sessions and we cannot retrieve the Spark session to issue the Spark tasks to the Celery worker. At last we use getOrCreate to point to or create the Spark context and Spark session by loading the sparkconf_builder of the spark_celery_app.

Results

Outcomes: We built recommendation system targeted on Yelp new users and Yelp existing users respectively. For the recommendation system for Yelp new users, the system outputs the result that match with the input keywords most and also recommend the restaurants with high sentiment score that have the most positive reviews. For the recommendation system for existing users, we built a hybrid recommendation system that is combined with item-item similarity, user-item similarity, and friends social network methods. For item-item similarity, we output restaurants that match with the restaurants visited by the user before. For user-item similarity, we use collaborative filtering method and predict user interests by collecting preferences information from many users. For friends' social network, we see friends' user id from each user and groupby and count the restaurants with top rating given by user's friends. Then, we output the top restaurants that have most count of top rating given by their friends.

Lessons from Data Analysis: We learn that we need to know each feature well and how each feature correlates with each other before we start doing data analysis. After getting to know the background of data, we generally have clear ideas on data filtering and use the clean datasets to produce result for the project. There are several valuable experiences we gain from 'data analysis'. First, we need to decide which plot to produce for certain features. Secondly, we learn how to split the features with list and dictionary type using python regular expression, so we can further analyze the data. Last but not least, we learn to clean datasets for data analysis using pyspark dataframe, RDD, and SQL methods. Since Yelp dataset is quite large around 3GB with compressed data, pyspark would be a great tool for data preprocessing.

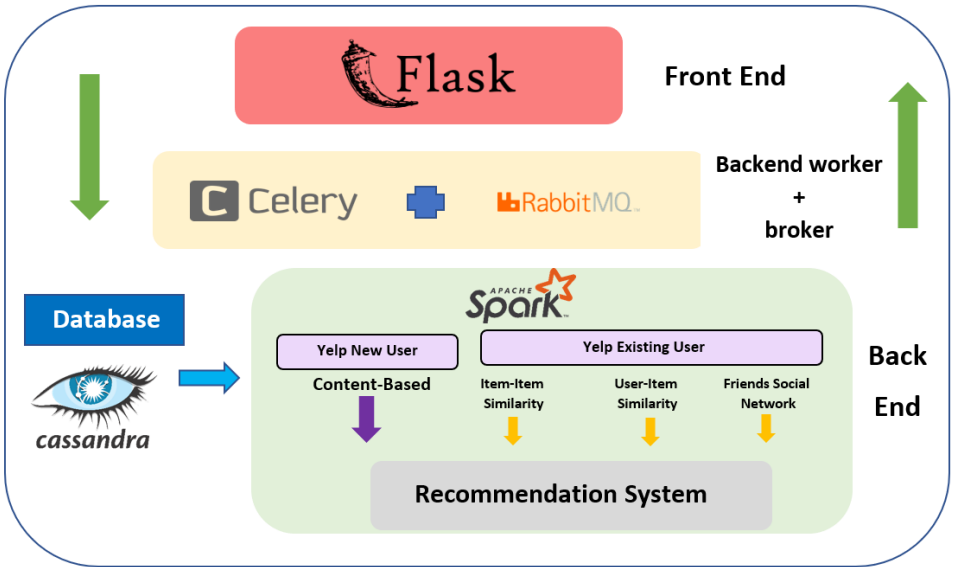
Lessons from Flask Frontend Implementation: We now learn how to effectively generate frontend web pages based on data received from cloud computation. We have learnt how to host websites connecting the web application with the cloud. We understand the job nature of the broker as a communicator to keep track of jobs Celery runs at the background and delivers results to the Flask app. We know how to update the web pages based on data received at any time. We know how to receive data from the webforms posted, and to issue Celery tasks based on the query. If we are to build another Flask application, we can do it much more efficiently as we are familiar with how to build one.

Project Summary

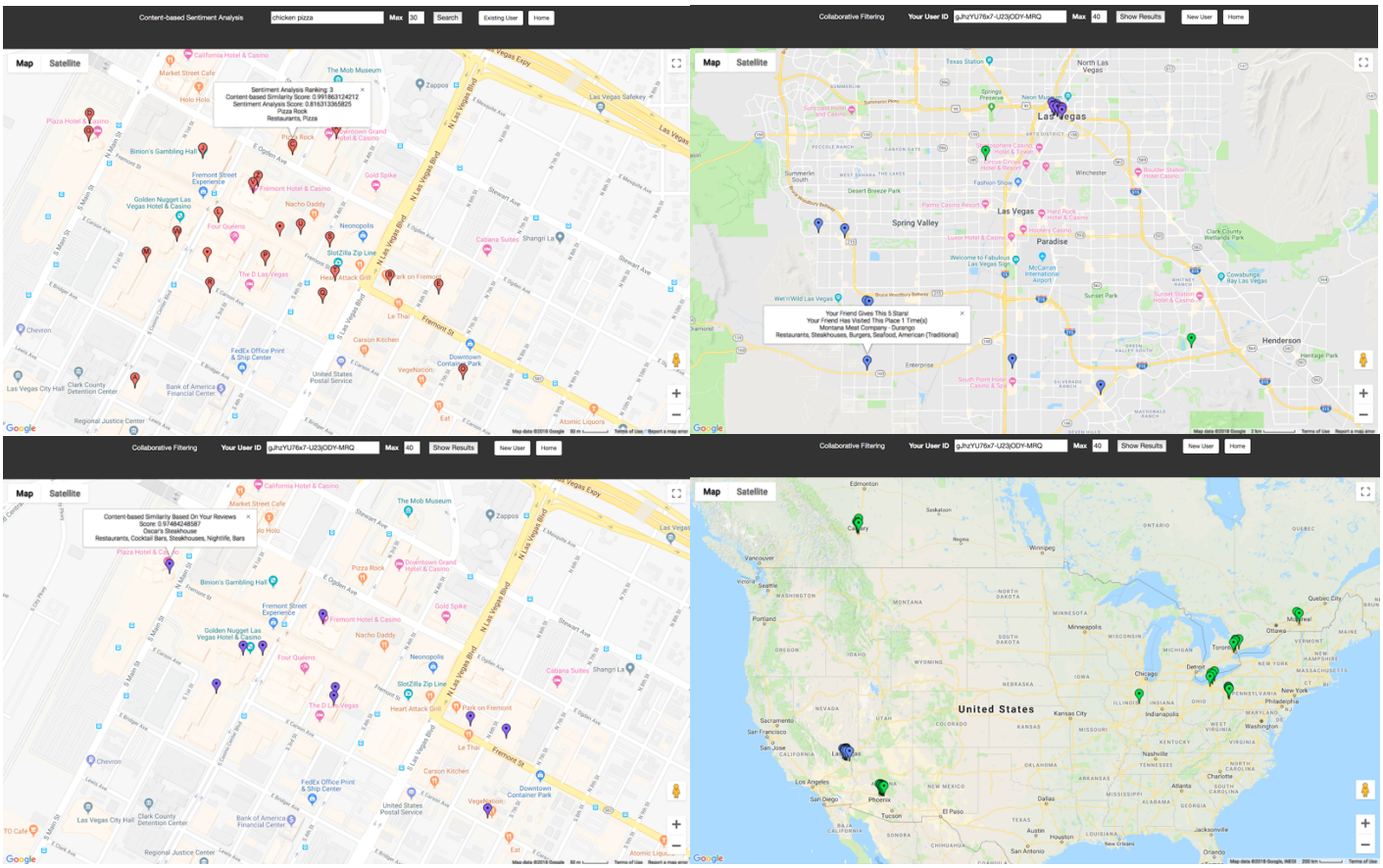
Categories	Points	Reasons
Getting the data	-	Use open datasets from Yelp Challenge.
ETL	2.5	Filter the business data through latitude and longitude. Split category features from list and get popular categories. Split attributes features from dictionary type. Split hour columns into separated columns. Split check-in time features into separate columns.
Problem	1	Identify problems: limited search fields on Yelp, deception of high rating, lack of features to predict users' preferences based on users' data.
Algorithmic work	5	Compare Word2Vec and TFIDF-LDA similarity score through cosine similarity formula. Train Keras Convolutional Neural Network to predict sentiment score between 0 and 1. Apply Word2Vec for item-item similarity method. Train ALS model for user-item similarity method. Split user friends from user dataset and seek for top rated restaurants given by their friends. Process Spark tasks and communicate between Flask frontend and workers with Celery.
Bigness/parallelization	1	Yelp data size is about 7GB. It takes one to two minutes to finish running the prediction of all three types of filterings given a user ID. It takes ten to twenty seconds to calculate results of content-based search.
UI	5	We build and generate a web application using Flask templates based on data received, build the interactable webforms and create customized Google Maps. We add Bootstraps design.
Visualization	2.5	We use Tableau to visualize the characteristics of data collection on maps, social network graph, and charts.
Technologies	3	Spark, Keras, Gensim, Flask, Celery, RabbitMQ, Tableau

Illustrations

System Architecture:



Content-based Sentiment Analysis (Top Left Picture): Collaborative Filtering (Other Pictures):



References

Celery-Spark (Provided by Professor Gregory Baker): https://github.com/gregbaker/spark-celery
WTForms: https://wtforms.readthedocs.io/en/stable/
Flask-GoogleMaps: https://github.com/rochacbruno/Flask-GoogleMaps