Project Milestone

The Restaurant Dilemma: Personalized Recommendations for Groups of People

Have you ever wanted to find a restaurant for you and a friend to go to, only to have trouble finding one due to conflicting tastes and tedious perusing of restaurant listings and reviews? Wouldn't it be so much nicer if you could get recommendations based on both of your preferences?

Aim is to use Yelp data in conjunction with machine learning to find a restaurant which will suit the tastes of group of two or more people.

My deliverables for this project will be a front end app in flask with the back end algorithm in python to provide recommendations.

In this project I plan to use collaborative filtering method to build the recommendation engine.

The collaborative filtering allows to make tailored recommendation for individual users, even if the particular user's taste of dining is not fully reflected in her/his Yelp review history.

Also there will be a component in the system to give more weighing to the recommended restaurants reviewed by people in a users social network followed by the weighted to trustworthy reviews if the recommended restaurant is not reviewed by any of the friends in the social network of the user which builds trustworthiness of the reviews.

1. Data Source:

The source of our data is courtesy of the Yelp dataset challenge. The purpose of the challenge is to use the provided data to produce innovative and creative insights.

Included in the dataset were five json files/sql file, which are encompassed users, checkins, tips, reviews, and businesses. Yelp also included 200,000 photos with their associated labels, though we did not incorporate any image analysis in our end product. The total size of the text data was roughly 5Gb. While this barely qualifies as 'big data,' our desire to use big data tools and techniques is justified. The richness and variety of the data gave us the freedom to apply a wide range of machine learning techniques to create what we called “Yelper Helper.” The goal of Yelper Helper was to build a recommendation app for users based on keyword inputs, location, social networks, and reviews. This will be explained in more detail in the following sections.

2. Data Preperation:

First I Imported the self contained sql file in mysql database and started with cleaning the data.
After going through business table I realized this dataset contains information on all the business . So I cleaned the dataset by extracting businesses which are associated with food or restaurants category in category table.

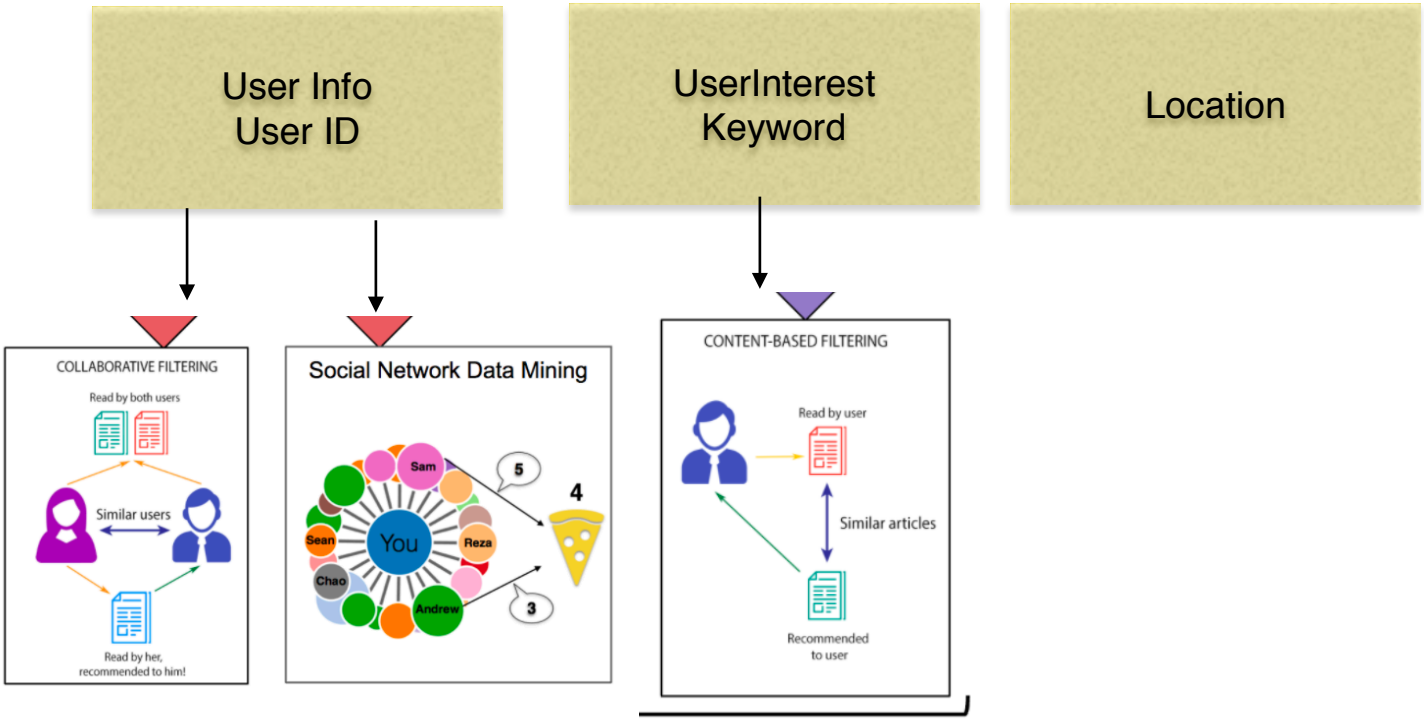
Also this dataset contains data going back to 2004 which I felt is irrelevant now and is also increasing the data size while doing analysis in pandas data frame which increases the computation time and cost so I deleted data from all the tables for time frame between 2004 to 2012. By doing this it reduced the size of the data drastically from 5gb to 2.1 gb

Reason for my sql:

MySQL provides many advantages for my recommendation app. First, some of the data files contained millions of rows, so I spun up the MySQL instance and loaded each dataset into separate tables. This made data extraction easy and fast, depending on what I want to analyze. Initially, eight tables were created, one for reviews, businesses, tips, check-ins, users, attributes, elite users. A visualization of schema can be seen here:

An additional advantage MySQL provided was that it allowed us to easily establish a connection via sqlalchemy and MySQLdb to Spark and Python, thereby making it unnecessary to create multiple intermediate/temporary csv files.

3. The Recommender Engine :

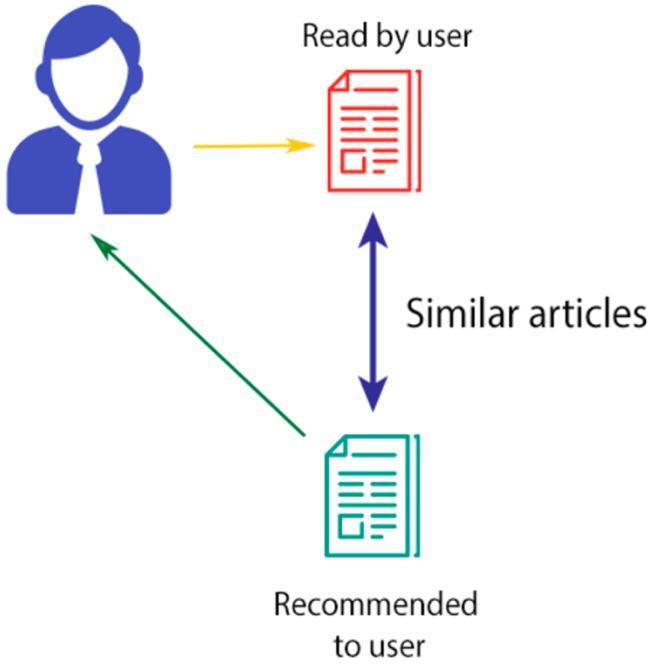


3.1 Natural Language Processing (NLP)

3.1.1 Sentiment Analysis

First, I have decided to analyze the Arizona subset using Natural Language Processing (NLP), a machine learning technique that aims to understand human language with all its intricacies and nuances. My goal is to create a supervised learning neural network that could predict the sentiment of a review as positive or negative based on the language used.

3.1.2 Content-Based Recommendations

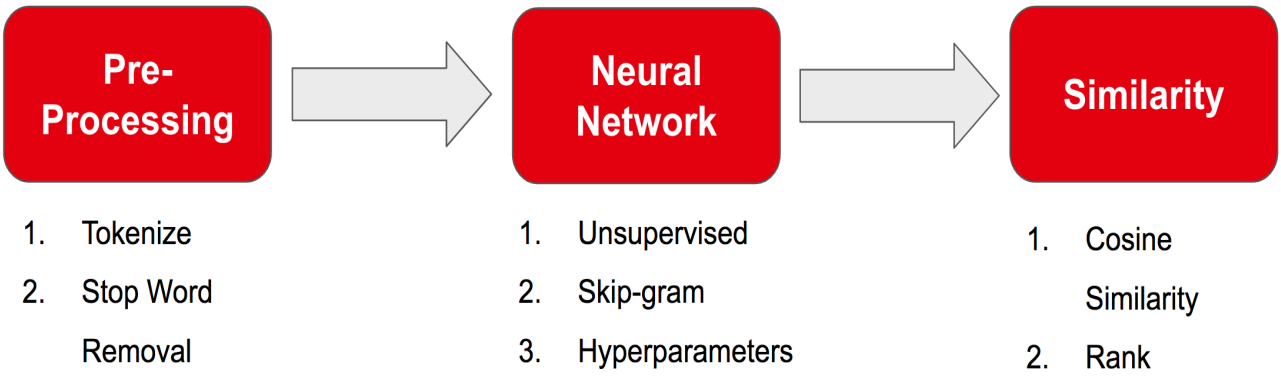


I plan to use NLP to make content-based recommendations. Item-to-item similarity is calculated to make these types of recommendations. Think of when you read an article online: there is usually a side menu or a section at the bottom of the page that recommends new articles based on your interest in the current article. Often, these new articles are written on the same or a similar topic. That's a content-based recommendation.

Content-based recommenders can take a user's profile and past ratings to make new recommendations to the user. This, however, presents the cold start problem, an issue that arises for a brand-new user who has no rating history. To combat this, my NLP recommendation is based on a keyword soft match (similarity calculation) rather than the

user profile. So, typing “tacos” should bring up a ranking of Mexican joints, with the possibility to also recommend some non-Mexican places serving taco-like food.

To make this happen, once again I plan to implement a neural network to understand the language used in the reviews. I plan to use the Word2Vec function from Spark MLlib to create the model. Our process is outlined in the figure below.



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 farther apart. For example, the vectors for “ice cream” and “frozen yogurt” should be pointing in nearly the same direction, but the vectors for “delicious” and “disgusting” would be far apart.

Once the words are translated into vectors, in order to check if the result makes sense (and have some fun), we can perform some word algebra. We can add or subtract the vectors from one another to find a new word.

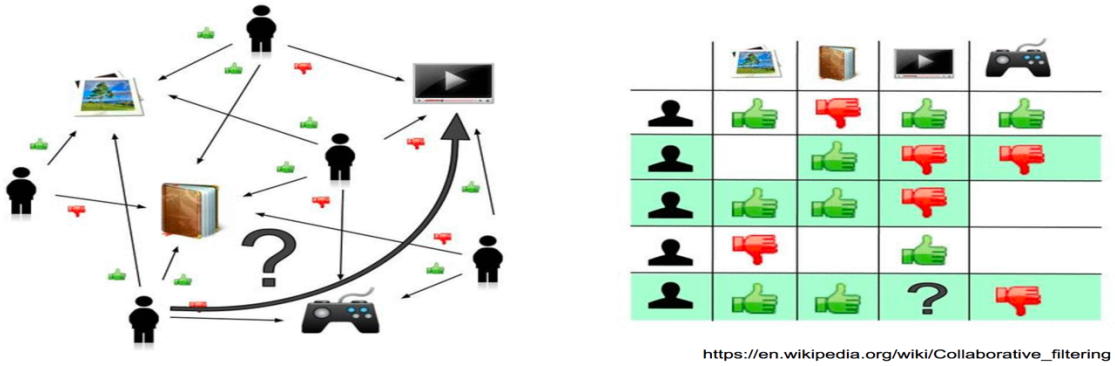
In addition to using word algebra, we can determine how similar two words or documents are based on cosine similarity. Mathematically, cosine similarity is the dot product of the two vectors divided by the product of the magnitudes of those two vectors. This calculates the angle between two vectors. The smaller the angle, the more similar the words, the closer to 1 the value becomes. Larger angles are more dissimilar and are calculated closer to negative one. Vectors perpendicular to one another will have a cosine similarity of zero.

3.2 Collaborative Filtering

Collaborative filtering (CF) is commonly used for recommender systems. These techniques aim to predict user interests by collecting preferences or taste information from many users. In other words, CF fills in the missing entries of a user-item association matrix. The

underlying assumption is that if person A agrees with person B on one issue, A is more likely to have B's opinion on another issue than that of a randomly chosen person.

Below is a great visual example from Wikipedia. In order to predict the unknown rating marked with '?', we rely more on the opinions from other users with *similar* rating histories (green rows), thereby arriving at a negative rating prediction (thumb's down).



https://en.wikipedia.org/wiki/Collaborative_filtering

Mathematically, this is done by low-rank matrix factorization, combined with a minimization problem (see picture below). The often-sparse user-item rating matrix R is approximated as a product of user matrix U and item matrix PT, which are built of latent factors. We then form the cost function J, and try to minimize it. Currently in the *spark.ml* library, the alternating least squares (ALS) algorithm has been implemented to learn these latent factors. Additionally, since we directly rely on the user rating itself, our approach is often referred as "explicit."

3.3 Social Network

This is a digital version of the classic Word-of-Mouth recommender system -- what people have been using for thousands of years.

With the social network database now structured and easily searchable, a few SQL join-groupby-aggregate commands can quickly answer questions like: "What are the average ratings of the nearby restaurants *only based on my friends' opinions*?" This interesting feature has the potential to both provide conversation-triggering recommendation and improve user stickiness to the app. As an extension of the algorithm, one can easily come up with other intuitive rating estimation schemes, such as expanding the network to second- and third-degree connections and applying a weighted average.

Users who come to our webpage may want a quick suggestion of all possible restaurants based on only their location.

While yelp provides aggregated ratings for each business, these are not always indicative of a restaurant's quality. For example, a restaurant with one five-star rating would be ranked ahead of a restaurant with ten ratings averaging 4.9 stars. Another problem is that a star rating varies from person to person and is integer based.

My strategy for dealing with these problems is as follows:

- On the review level, apply a time weight and a sentiment weight to get an up to date, accurate representation of each review

On the business level, modify the adjusted star rating with popularity features Based on the NLP sentiment analysis described above, each review is assigned a sentiment value. This is indicative of how positive a review is with a range of zero to one

On the business level, we also need to address the popularity measures: review and check-in counts. A high review count is indicative of either a popular business or an exceptional one. Check-ins are a more direct measure of popularity but it is a more recent feature so even the most popular businesses have counts around 160. With this in mind, check-ins are assigned approximated twice the weight of reviews.