Final Project Report

"We Like Food"

Yuqian Gong (Nancy), Leoson Hoay, Lily Li, Alex Maiorella

Dataset:

Yelp Academic Dataset (6.5GB)

- 174k businesses
- 5.2 million reviews
- Dated 2004 - 2015
- 1.3 million users
- 11 metropolitan areas

Research Objectives:

1. Build a pair-based restaurant recommendation system.
2. Do businesses perform better when they are near other businesses, or when they are far away?

1. Restaurant Recommendations:

The foundation of the recommendation system is built on between-user similarity and between-restaurant similarity. In computing these values, we draw upon the densest source of information provided by the Yelp dataset - the review text. Between-user comparisons were made with this in mind, while between-restaurant comparisons also included the element of distance, given that distance is likely to be a strong factor of consideration when visiting an eatery. In order to avoid directly pairing millions of reviews and making the process improbably difficult to run, we use the between-user pairing to find each user's most similar partner, which serves as the preamble for recommending restaurants.

We make document comparisons largely with the assistance of the Gensim Python library, which facilitates the task of building a document-to-vector model. Each document is transformed into a vector based on a Latent Semantic Analysis model, and cosine similarity is used to calculate the similarity between document vectors.

Because of the exceedingly large number of data points, running the dataset and pairing it can become an exponentially intensive process (imagine 1.2 millions users/5.2 million reviews squared). Training the dictionary on a set of 1 million lines of reviews took approximately 30 minutes running 7 Google Dataproc standard cores. The granular LSI model based on 1 million data points also required approximately 2GB of RAM. To get around this we sampled chunks from our data, and we also made use of the n1-highmem cluster instances to boost the memory of the worker cores on the cloud (*--instance-type n1-highmem-2*).

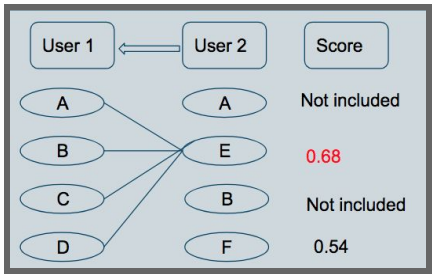
Mapreduce with Google Dataproc was used to facilitate the steps in this process. In the first step, we produce a dictionary from review text.

```
{'accepts': 0, 'add': 1, 'apples': 2, 'arrived': 3, 'aside': 4, 'banchans': 5, 'beef': 6,
'better': 7, 'bigger': 8, 'bite': 9, 'bottomless': 10, 'bowl': 11, 'centre': 12, 'cheap': 1, ...}
```

This dictionary is then used to initialize a corpus and an LSI model, which maps the distribution of words within a subset of the data.

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 2), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11,
1), (12, 1), (13, 1), (14, 1), (15, 4), (16, 1), (17, 1), (18, 1), (19, 1), (2z0, 1), (21, 1),
(22, 3), ... ]]
```

With this, we can now compute similarities between reviews at the user level and the at restaurant level. For each user, we compare all the non-overlapping restaurants of their most similar user, taking the average cosine similarity and average haversine distance into account. A graphical representation of this process is as follows:



A snippet of what is produced by the final output is shown below:

```
["-0x2ov-qcCopv32Imm-TYg", "Oz21pBYHtf1DkcqmcUhhAQ"]      [0.17099509767516632, "EsMcGiZaQuG1OOvL9iUFug"]
["-hietrA8M58asfpyJkCyiA", "NwD3wqGB4RhAMVRy27duQQ"]      [0.1246109726957239, "BnuzcebyB1AfxH0kjNWqSg"]
["-ijJvthn69xBeTD6YAJ7wA", "rjhqh2eZpvl5uXZczsRtCw"]      [0.16674092014771882, "VXH7zXcZzXlmAVN8GSjGRQ"]
["00y_qiDGICaNqFZqxM-DfQ", "D2WVqRftDNqqBrRPpmghVA"]      [0.06985876251140821, "n7V4cD-KqqE3OXk0irJTyA"]
["0LMHrFTQtUoY42AZl1MH6g", "SlhGO5AKkJS3X8uw5Z9QQg"]      [0.15493739861715558, "PfOCPjBrlQAnz__NXj9h_w"]
["0l3x4EgoWVYdla3CPpAKag", "oi6A5zmKXA0wyhje0brnmQ"]      [0.16674092014771882, "PfOCPjBrlQAnz__NXj9h_w"]
```

The strings in the left tuple represent the user being recommended to and the user the recommendation comes from (the most similar user), respectively, and the right tuple represents the ‘recommendability’ score of the highest scoring restaurant, and the restaurant id, respectively.

To compute this score, we use the following formula:

$$Score_{rec} = \frac{\sum sim_{res}}{n_{ij}} \times \frac{1}{\log \frac{\sum Dist_{mn}}{n_{ij}}}$$

where i = a single user and j = the user most similar to i

We weight the average similarity scores of all non-overlapping restaurants from user j using the inverse logarithm of the average haversine distances. This is based on the assumption that the further a restaurant is from a user's general hangout radius, the less likely the user will be tempted to check out the restaurant, even if the restaurant might be within the user's preferences. The logarithm was taken on the haversine distance to suppress the large values and possibly large variance in the raw numbers.

2. Business Success:

For this question, we explored if businesses do better when they located near other businesses or when they are located far away from other businesses. Perhaps centralized locations of shops (ex. shopping malls) bring more clients to each store, or perhaps they draw away clients to a business's competitors.

We first used the VADER (Valence Aware Dictionary for sEntiment Reasoning) package to synthesize a numerical score for each business review. We then computed the average sentiment score for each business. Both steps were done using MapReduce in Google Cloud. We then used a SQL database to join several csvs, each containing different attributes of a business, into a "master" file, in order to expedite our final "similarity score" and "success score" computation. We combined all the csvs to avoid the computationally expensive task of searching for each business id in multiple files in MapReduce.

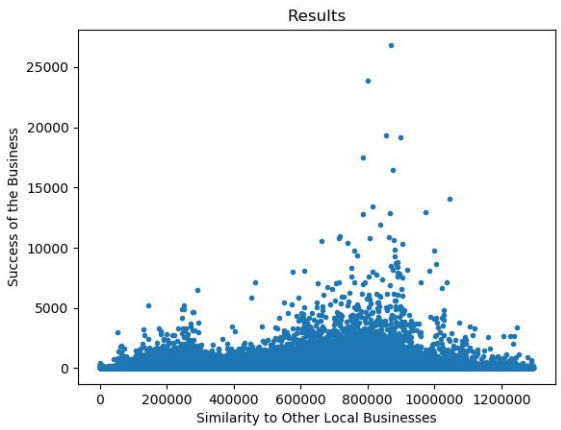
We then used MapReduce to calculate a weighted average of several factors to produce measurements for business success and similarity to other local businesses across the entire dataset. Pairwise similarity scores were calculated by pairing each business with every other business in the dataset. The algorithms are given below. This was executed in Google Cloud. Without cloud computing, the extrapolated runtime on our personal machines would have been 6 months.

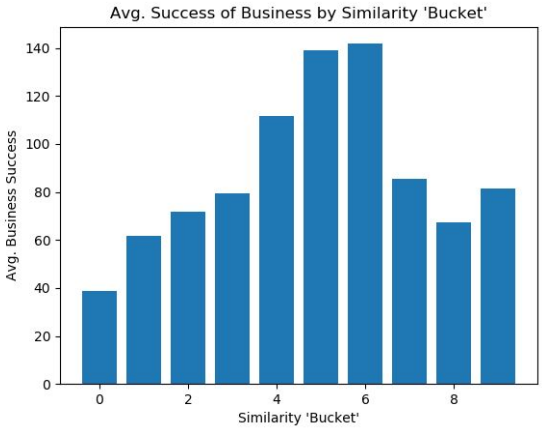
- **Success score:**
  (Average star rating) x (number of reviews) x (average sentiment score of reviews)
- **Similarity score:**
  Sum of "pairwise similarity" across other businesses within 50 miles
    - **Pairwise similarity:** (inverse exponential distance) x (category similarity ratio) x (rating similarity ratio)

Conclusions:

We graphed success scores vs. similarity scores to answer our initial question. A simple scatterplot was too noisy, so we binned the similarity scores and calculated the average success score within each bin to produce the graph on the right. From the graph on the right, we see that similarity and distance to other businesses appear predictive of business success. There is an

ideal balance between mutual benefit and competition. This ‘sweet spot’ emphasizes mutual benefit (think: Kimbark Plaza or Harper Court). Mutual benefit seems to be stronger than competition, as the average success scores are left-skewed. However, at some point, too many similar businesses leads to lowered as success, as the market becomes too saturated and competition wins out.





Challenges:

These are some of the challenges we faced across both of our questions/analyses:

- mrjob could not handle newline characters in reviews text
    - The mapper would read newline characters in the reviews string as a new line in the file, thus ignoring the actual column breaks. To resolve this issue, we had to first run clean_linebreaks.py on the reviews dataset to replace line breaks with spaces.
- Importing packages and NLTK/gensim corpus in mrjob
    - We faced difficulties importing packages with dependencies, like scipy, and packages with external corpuses that needed to be downloaded, like NLTK and gensim. We resolved this issue using bootstrapping in two manners: by using the mrjob.conf file (/mrjob_config/mrjob.conf) and bootstrapping directly in the terminal (/BusinessSuccess/command_line_mrjob_bootstrapping.txt)
- Trouble running MapReduce on whole dataset
    - Our dataset was prohibitively large, especially when it came to pairing. We solved this issue by upping our Google Cloud quotas significantly (to 125 cores) and splitting our dataset up into chunks when possible. We ended up pairing chunks of our data paired against the whole dataset and ran multiple jobs to analyze the whole dataset.