W.P.I. DS501; Introduction to Data Science

To: Dr R Paffenroth, Dr. X Kong

From: Diana Batista, Matthew Curcio, Marcus Moyses, Tabassum Kakar,

Chong Zhou (Andy)

Due Date: December 7, 2014

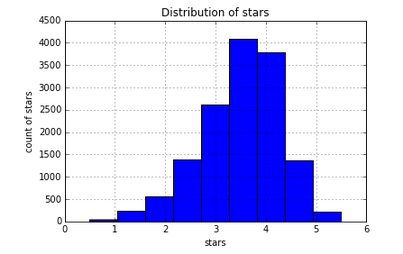
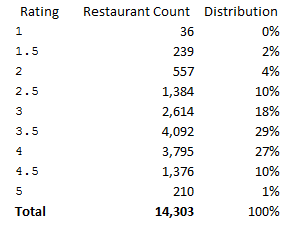
Subject: Case Study #4: Yelp Dataset Challenge

Introduction

In this case study we are analyzing Yelp data. This dataset contains 42K businesses, 253K users, and 1.1M reviews from Phoenix, Las Vegas, Madison, Waterloo and Edinburgh.

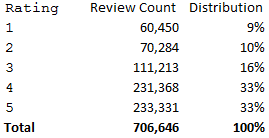
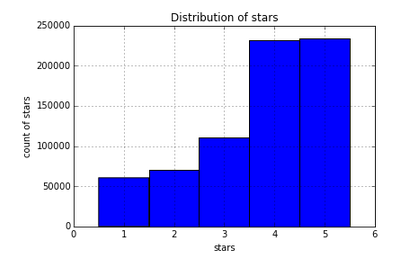
Part 1

We chose to analyze Restaurant data. There is a total of 14,303 restaurants collected in the Yelp dataset. A histogram of the distribution of ratings is as follows:



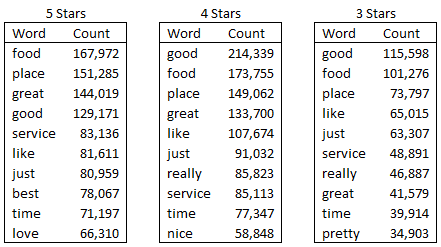
The rating distribution appears to be approximately normally distributed with a slight left-skew and thinner tails, i.e. platykurtic. We can see that 73% of the restaurants have a rating between 3 and 4, inclusively.

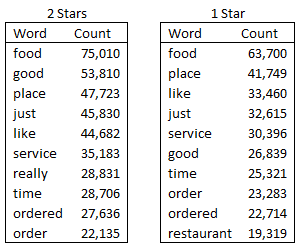
These 14,303 restaurants have a total of 706,646 reviews. Although the distribution of ratings among restaurants has an approximately normal distribution, the distribution of ratings among the actual reviews is quite different.

This information suggests that users are more likely to review restaurants when they have had positive experiences.

Next we analyzed the top 10 words present among each review rating using a max\_df of 0.8.

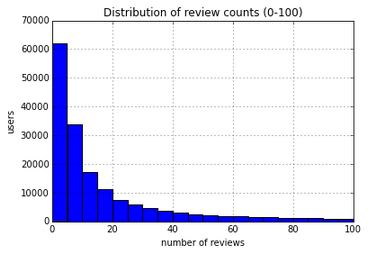
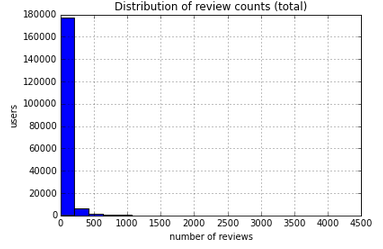




All reviews included the words “food,” “place,” “good,” “service,” “like,” “just,” and “time,” which suggests users are basing their ratings on similar criteria. The words “best” and “love” are unique to 5 star reviews while “great” is present in reviews between 3 and 5.

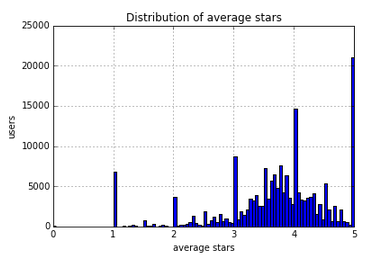
The 706,646 reviews were completed by a total of 185,469 users.

The review count for the user set is as follows:



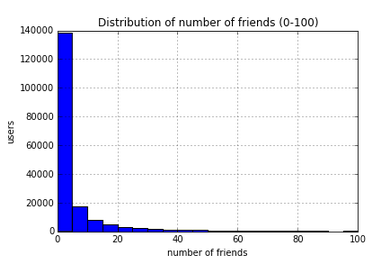
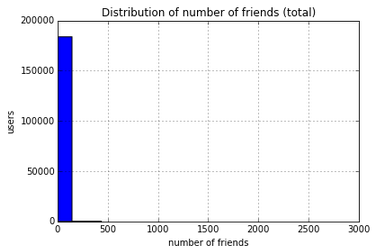
The first histogram shows that almost all users have rated restaurants less than 250 times. A closer look among users with less than 100 reviews shows that most users have rated restaurants less than 5 times.

The distribution of star ratings among users is seen below.



The overall trend is skewed left, similar to the count of restaurant by average rating. The spikes at the integers likely represent users who have reviewed only once. Ignoring these spikes, it appears the mean average rating is approximately 3.75. In future analysis, it would be advantageous to remove all restaurant ratings with n=1 reviews.

In order to analyze the distribution of number of friends, the data had to be manipulated in order to become manageable. It should be noted that the term “friend” is similar to that of a Twitter follower, meaning that a user can be a friend someone without that friendship being returned. Therefore, those users with no friends were removed and users who were “friends” with the removed subset were also excluded. The results are as follows.



Of the users that remained, most had less than approximately 125 friends. Zooming into the subset of users with less than 100 friends, it appears that most have between 1 and 5 friends.

Part 2

To further analyze the data set it was decided to use MapReduce. MapReduce allows programmers to parcel out the data into smaller more manageable groupings. This is generally thought of as a two-step process.

This first step is known as 'mapping.' It takes its meaning from the lisp language set commands that map resources to a problem for more efficient computer usage. Data is broken down into small subsets. A routine that then maps this subset of data to another process. This process can be all within one machine or mapped to multiple processors for subsequent calculations. It is easy to visualize that a smaller set of data can be worked on more quickly than its entirety. If many computer processors or resources are allocated to a project the work can be implemented very quickly overall.

Once each of the many calculations from the smaller sets are completed they are recombined. This part of the process is known as 'reducing.' This process can be on one computer or on many. The reduction takes the output from the mapped calculations and knits them together to obtain the results as if it were calculated as whole.

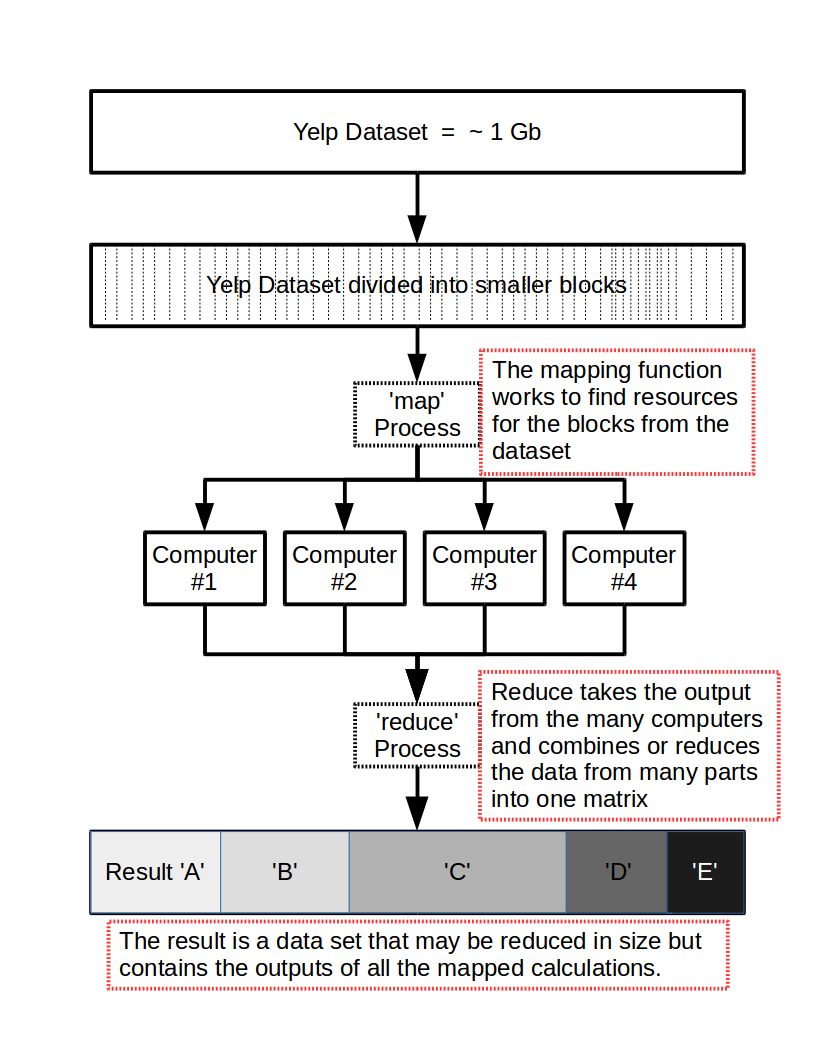
To start this process we must convert the data set texts files into a format usable with MapReduce, i.e. TFIDF format. 'Term Frequency - Inverse Document Frequency' (TFIDF) is a common method to transform raw text data into term frequency vectors. The algorithm is actually quite simple, as is shown below.



Where: |D| represents the total number of documents in the corpus, and

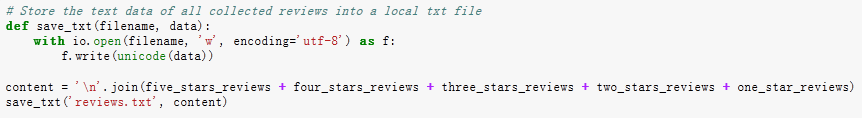
|{d: ti є d}| represents the number of documents where a term (denoted by 'ti') appears. The log of the ratio that these two values produce is useful because it reduces the amount of noise caused by spurious data points. The log ensures that one term must be seen many times for it to be considered valid.

The MapReduce procedure can be found in more detail on the last page.



This analysis was completed through the steps below.

1. In order to store the test data of all collected reviews in a local txt file where each line of the file contains the test of one review, we used an open function in the io package to open a stream which points out a file name and uses the write function to store the data into a file.



Next, we converted the file into TFIDF format using MapReduce. The TFIDF method is a way to convert text data to term frequency. Each term that occurs within the document is counted.

1. For the black box analysis, the input is text while the output is a matrix. The columns represent all terms that appear within the document and the rows represent each individual document. The numbers in the matrix indicate which words appear in the document.
2. In order to accomplish this, several smaller must be complete:

* Number of times a term (e.g. “good”) appears in a given document
* Number or terms (e.g. “good”) in each document
* Number of documents in which a term (e.g. “good”) appears
* Total number of documents

1. Depending on the task done through the map-reduce system, we could not implement the TFIDF method in one map-reduce job. Two steps were needed in order for the calculation to be complete.

* Job1: for the Word frequency in each document

Mapper:

Input: (document\_name, lines)

Output: (word, 1)

Reducer:

Sums counts for word in document

Outputs ((word, document\_name), n)

* Job2: For Word Counts For Docs

Mapper:

Input: ((word, docname), n)

Output: (docname, (word, n))

Reducer:

Sums frequency of individual n’s in same doc

Feeds original data through

Outputs ((word, docname), (n, N))

* Job3: Word Frequency In Corpus

Mapper:

Input: ((word, docname), (n, N))

Output: (word, (docname, n, N, 1))

Reducer:

Sums counts for word in corpus

Outputs ((word, docname), (n, N, m))

* Job 4: Calculate TF-IDF

Mapper:

Input: ((word, docname), (n, N, m))

Output ((word, docname), TF\*IDF)

Reducer:

Just the identity function

1. Implement every job.

When implementing the jobs, there are some things that need to be considered.

* Each review is in one line, which reduces some of the work for us since the data is pre-procedured.
* Pipe line can be used, where every step takes the result of the previous task as input.
* Job 3 and Job 4 can be done in one reducer, which will save some i/o cost.

Our implement method is:

* Job1:

Mapper:

Input:( word\_key , line)

Output:

Total number of documents:(None, 1)

Frequency on review (word, 1)

Reducer:

Input: (key , values)

Output:

Total number of documents (None sum(values))

Total count in review (None, (key, words\_counts))

Number of documents a word appears

(None,(key,sum(values)))

* Job2:

Reducer:

Input(\_,values)

Calculate tf-idf value

Tf= values

Idf= log(total number of words / word\_counts for each word )

Output: (i , (word , Tf\*Idf ))

Job 1 is to calculate all the information needed, while Job 2 is to converge the information into a TFIDF matrix.

In the last output, “i” means the lines (which indicate the documents), word, and their TFIDF value. It is a large sparse matrix. Presenting data in this way (line, words, tf-idf value) will save space.

1. In order to make coding easier, we did not use Combiner.

Part 3

The original Yelp database used in this exercise had 252,898 users. However in our data set many uses had zero or no friends, therefore if we created a graph representing the connections between users we would end up with many nodes which had no out links. Nodes that do not link with others are called 'sinks' in the page rank algorithm, since these nodes do not pass forward their page rank. Since sinks detract from the page rank these do not contribute positively or otherwise to this groups work.

The first attempt to create a page ranking was greatly hindered by the fact that there were too many sinks. This produced a large sparse matrix that could not completed. After realizing this problem a calculation was made to estimate the size of the graph. Compiling all users into the page rank algorithm would require a sparse matrix with approximately 3.276 x 10^9 non-zero entries. This number was too large for our personal computers.

Alternatively, it was decided to trim users that had zero and one friend(s) from the data set, since those users have little or no contribution in the overall page rank score. After removing those users we ended up with 92,963 users or nodes. This value is a more reasonable set, but since we removed nodes from the graph we had to make sure no other node would have connections to those nodes that were removed, so a second trimming was undertaken remove those inconsistencies. The final number of nodes we were left with was 92,817 users. This reduction represents a 36% drop from the initial number of 252,898 users and left us with more manageable run times (on the order of 4 hours) on relatively small laptops.

From this reduced set of users, a sparse matrix was created which represented the connections in the graph. This new matrix was then multiplied by a vector of size 92817 that would hold the page rank scores according to the formula, **It . H = It+1**.

This set of matrix multiplications was reiterated through until it converged asymptotically.



Table #1: Page rank scores for the top 30 users.

Table #1 shows the page rank score of the top 30 users in the data set from the 92817 users total after trimming. We can see that these top users have increased their page rank score. For comparison, if we investigate this further we find that each of the 92,817 users would initially have a page rank score of approximately 3.95 E-006. If we normalize the values in table #1, we would find the top user would have increased their score by 280%. The 30th top user increased their page rank scoring by over 228%. Meaning these users wield more than twice the influence compared to the average user.

If all users had equal page ranking then each user would be:

1 user / 92,817 users total = 3.95416333858E-006

The number 1 top user percent increase page rank score:

(1.108270255E-5 / 3.95416333858E-006) x 100% = 280.3%

The 30th top user increased their score by:

(9.0454471419E-6 / 3.95416333858E-006) x 100% = 228.8%

Part 4

The final question investigates what would happen if the rating of the businesses was not simply the average of the rating of the reviews but included the page rank of the visiting users. As we have seen above a small group of users have an increased amount of influence compared to others. This could be due the fact that these select users are more 'famous' or their ratings are listened to by many more people than the average user. There are many possible explanations for this. For example their views are more respected than others or they are knowledgeable in some areas than others. Including page rank information can greatly affect how other users perceive that business (in our case restaurant) therefore we computed the ratings of the businesses in the data set using the new calculated page rank score of the users as a weight for their reviews. The mean restaurant scores are now weighted means.



Table #2: Restaurant scores before and after using a weighted mean.

Table #2 shows the top 20 businesses which had the largest difference in their ratings, considering the absolute value of the difference, for example, a 3.0 star difference means that a business could have an average rating of 4 stars but dropped to 1 star after the weights were considered.

Many of the top 20 restaurants had at most 5 reviews and that's why their ratings changed so much. They probably had a very influential user among those few reviewers and their page rank score has a lot of weight, considerably changing the final rating of the business.

What is interesting is that dining establishments such as Denny's went down in the page rank star rating. This is also true of Jack-in-the-box as well as McDonalds. These are not known for high end dining and may indicate that people with clout actually do not spend their dining dollars on these franchises, but may go for restaurants where the rating did not fluctuate as much.

# References:

1. http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

2. Wikipedia

3. .<https://www.youtube.com/watch?v=g9p1ji4EFLc>