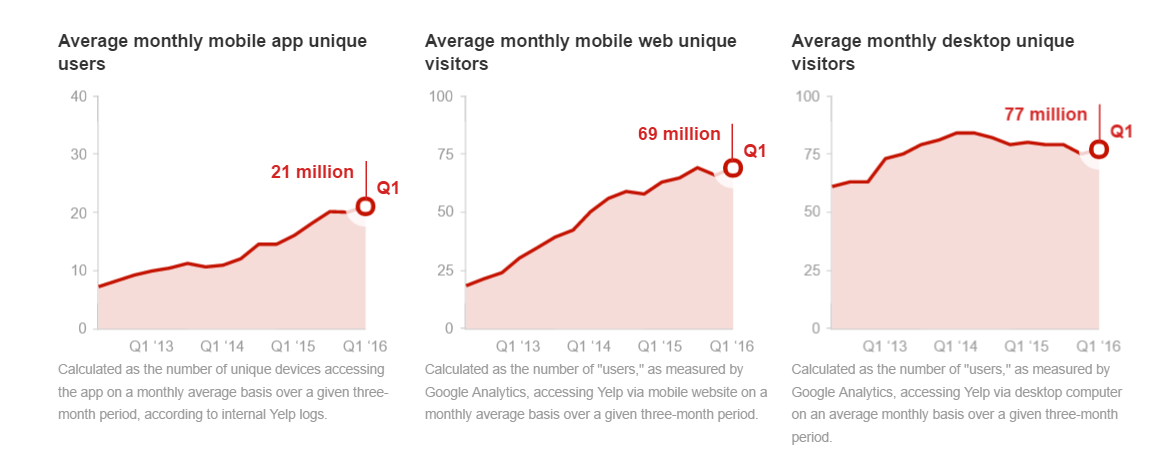
**Yelp Data Analysis**

*-Zhilong Hou, Jiawei Li, Li Ma, Nayyar Unda*

# Introduction

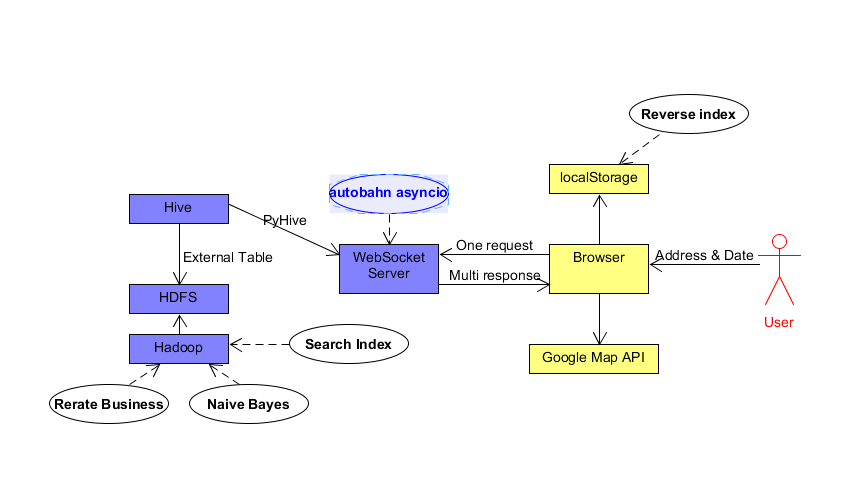
Yelp is a well-known website and mobile app which publish crowd-sourced reviews about local business.It has become very popular in recent years. Yelp metrics shows an average of 77 M desktop, 69 M mobile web and 21 M mobile app users visiting yelp website every month. But how many of these users actually review the website. How correct or reliable are their ratings and what’s the sentiment in their reviews? 

Our goal is to analyze 2.3 gb’s of user generated content provided by yelp dataset challenge. It has local data set from 10 cities across 4 countries. It has 2.2 M reviews and 552k users, 77k business and 200k pictures associated with it.

# Objective

Implement MapReduce based Big Data processing technique to analyze each user’s rating habit and re-rate each business, categorize each review by its sentiment analysis, study the seasonal trend based on the review count of each business,develop yelp’s functionality to find more popular businesses near by users and finalize the result in a well-designed web interface.

# Architecture



The architecture consists of map reducing phases for rerating the business , sentiment analysis using naive bayes, building a search index. Hive is used for querying the resulted data from HDFS. Pyhive will extract data from hive and send it to websocket server.websocket server will take one request and send multiple response based on users’ input. User can search based on the address, location or zip code.

Here are the technologies we are using

* Java,
* Javascript,
* Python,
* Hadoop,
* HDFS,
* Hive,
* PyHive,
* Websocket,
* Autobahn asyncio,
* Google Map API,
* Reverse Index

# Yelp Dataset

All the provided dataset are in six separate JSON files, we mainly use three of them, Business, Review and User(other dataset files are available on yelp data challenge), the schema of the raw dataset is shown below:

Business:

*{  
 'type': 'business',  
 'business\_id': (encrypted business id),  
 'name': (business name),  
 'neighborhoods': [(hood names)],  
 'full\_address': (localized address),  
 'city': (city),  
 'state': (state),  
 'latitude': latitude,  
 'longitude': longitude,  
 'stars': (star rating, rounded to half-stars),  
 'review\_count': review count,  
 'categories': [(localized category names)]  
 'open': True / False (corresponds to closed, not business hours),  
 'hours': {  
 (day\_of\_week): {  
 'open': (HH:MM),  
 'close': (HH:MM)  
 },  
 ...  
 },  
 'attributes': {  
 (attribute\_name): (attribute\_value),  
 ...  
 },  
}*

Review:

*{  
 'type': 'review',  
 'business\_id': (encrypted business id),  
 'user\_id': (encrypted user id),  
 'stars': (star rating, rounded to half-stars),  
 'text': (review text),  
 'date': (date, formatted like '2012-03-14'),  
 'votes': {(vote type): (count)},  
}*

User:

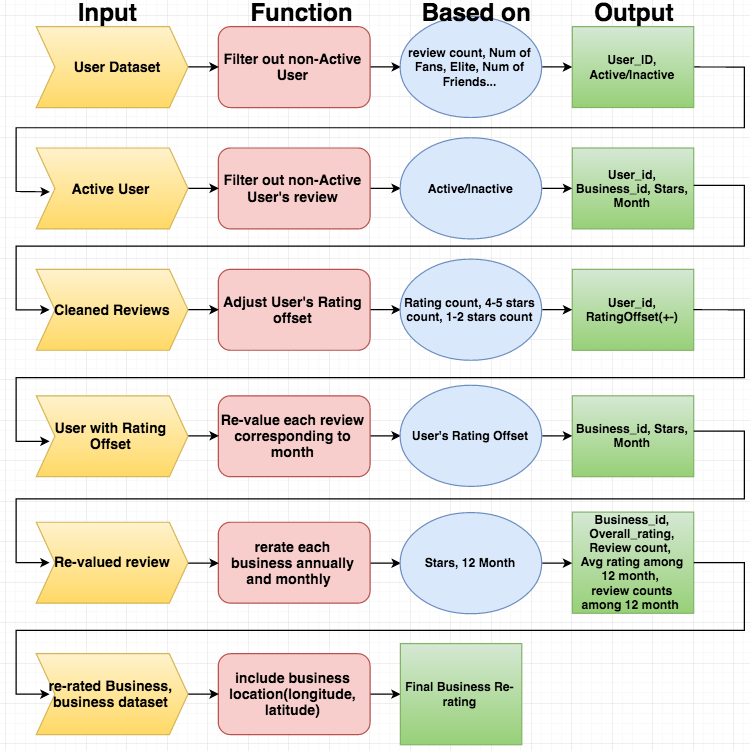
*{  
 'type': 'user',  
 'user\_id': (encrypted user id),  
 'name': (first name),  
 'review\_count': (review count),  
 'average\_stars': (floating point average, like 4.31),  
 'votes': {(vote type): (count)},  
 'friends': [(friend user\_ids)],  
 'elite': [(years\_elite)],  
 'yelping\_since': (date, formatted like '2012-03'),  
 'compliments': {  
 (compliment\_type): (num\_compliments\_of\_this\_type),  
 ...  
 },  
 'fans': (num\_fans),  
}*

# Re-rate and index Business

Based on the dataset provided by Yelp, the Businesses are scored by an overall rate which is basically the average rate given by all their customers. Such way of rating each business can be biased and needs to be adjusted to fit every user’s rating habit. We Assume that each user may have their own standard for rating businesses. It is reasonable to consider that some user may be generous and rate each business on high side, some users may be stingy of their stars and rate each business on low side, while others tend to be fair to the businesses and give neutral opinion to each one. So we should first analyze each user’s rating habit based on the review dataset, and then apply such result to re-rate each business.

Furthermore, when we are searching business on Yelp, it only shows the businesses nearby based on its location and performance. We think the seasonal effect of each business can be take into account as well. So after re-rating the business, we can also analyze the amount of reviews in each month and the average ratings in each month for every business so that the front end can show the most popular restaurant based on the date and location the user inputs.

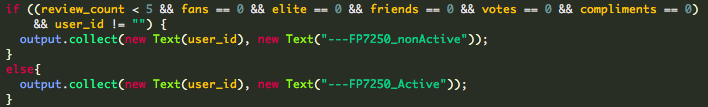
The whole data processing process is accomplished in six MapReduce jobs and deal with JSON format raw dataset. We ran the jobs in pseudo distributed mode so the outputs can be stored in HDFS for better implementation of HIVE for later stages. The following is the detail explanation of each MapReduce jobs:



*Mapreduce Flow Chart(6 steps of re-rating and indexing the businesses)*

All the raw dataset is in JSON format, so in MapReduce method, we are using Json.simple api to get JSONObject from the data.

(1). Filter out inactive User

When deal with big-data, it is important to do some data cleansing for the raw dataset. This saves a lot of resources and gets more accurate result. Before evaluate each user’s reviews and ratings, we think it is critical to filter out all the inactive users, so that we can only consider the active user’s reviews to each business. So according to the user’s dataset, we define a user as inactive if his/her number of reviews is less than 5, and has no record of key attributes like fans, elite, friends, votes and compliments.

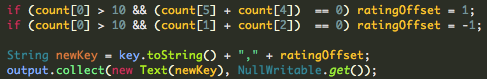
After running one mapreduce job, the output-key value pair output is userid, Active/inActive.

(2). Clean the review dataset

After classifying the users into active and inactive, the review dataset can be also cleaned. We can simply run one Mapreduce Job and filter out the inactive users’ review, the output key value pairs are business\_id, rating stars and month of the review for each review. So after this stage, we have done all the data cleansing, and having the valuable review dataset for active user waiting for re-value based on each user’s rating habit.

(3). Analyze User’s rating habit and adjust User’s rating offset

The idea of analyzing user’s rating habit is that we want to make the review of each user fairer to the businesses. For easier implementation, we only adjust the review scores of the users who have lots of reviews, and assume the rating offset +1 if the user is less likely to give high scores(4-5 stars), the rating offset -1 if the user is less likely to give low scores(1-2 stars). For the size of our dataset, we assume the user who has more than 10 reviews would have impact on the business rates if he is biased. The code snippet shows our judgement statement:



After running the mapreduce job based on the cleaned review data set, the output shows set of users and his/her rating offset. There might be more statistical solution of this, for instance, compare each user’s specific rating with all other users rating towards one specific business, then globally adjust each user’s rating, but those statistical method are complex to implement and our objective is only to process the data with big-data approach, from the perspective of the result, our implementation works fine.

(4). Revalue reviews

In the fourth stage, we are taking list of user’s rating offset and the cleaned review dataset as our input. Then we can re-value the reviews by adjusting each review’s rating stars. The mapreduce job takes the user rating offset, and adjust their rating stars in the business reviews scaling from 1 to 5. The output key/value pairs are Business\_id, star, month. We included the month of the review created as the preparation for the seasonal effect analysis and indexing for the later stages.

(5). Re-rate each Business based on revalued reviews

Since we have already revalued each review, it is time to gather all the reviews and re-rate each business. The review rating is much fairer after exclude the personal tendency of the user rate higher or lower than normal. Therefore, we takes seasonal effect for each business into account. Since last output includes month for each review. We can also know the number of reviews and the average ratings of the business for every month. The number of reviews reflects how popular this business is in this month and the average ratings shows the performance of the business during such month. So the output key value pair is as below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bus\_id | avg\_str | n\_rate\_t | avg\_Jan | num\_Jan | avg\_Feb | num\_Feb | ... | avg\_Dec | num\_Dec |

Where bus\_id is Business\_Id, avg\_star is the overall rating for that business, n\_rate\_t is number of reviews in total, avg\_Jan and num\_Jan … stands for pairs of average ratings among 12 months and number of reviews among 12 month.

(6). Build Business Index for querying

In our project plan, the result of our mapreduce job would finally be integrated with a front-end web interface. Such interface would take Month and Location as input and extract the result from HDFS to find the best place near the location during that month. So we take the raw business dataset as well as the re-rated business dataset as the new input, and add latitude and longitude fields into the re-rated business dataset, Furthermore, create a business index for detail informations for each business in JSON format as well so that we can easily query and parse our result in our front-end.

# Sentiment Analysis on Review Dataset(Naive Bayes)

Sentiment analysis generally aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be his or her judgement or evaluation, affective state or the intended emotional communication. In the Yelp review dataset, there are review text corresponding to the star ratings, which is a good source to do sentiment analysis.

After researched some online sources, we firstly looked into platform like NLTK(Natural Language Toolkit) and Stanford CoreNLP. They are all leading platforms for building programs to work with human language data. But they all require heavy dictionary to work with, which is considered as not a good implementation of Big Data technique. Then we found Naïve Bayes algorithm can be implemented in our case efficiently. Naïve Bayes algorithm is one of the most important supervised machine learning algorithms for classification, the classifier is a simple probabilistic classifier based on applying Bayes’ theorem. The Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables in a learning problem.

Firstly, we tried to run Mahout’s implementation of Naïve Bayes on our dataset, which turns out it consumes too much memory of our machine when computing the word matrix. Consequently, we decide to write our own MapReduce method to implement Naïve Bayes Algorithm to analyze our review dataset.

The Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes’ theorem as follows:

The classification has an assumption that attribute probabilities are independent given the class , where is attribute of the data instance. The assumption reduces the complexity of the problem to practical and can be solved easily.

For text classification, we should firstly calculate the probability of the sentence given as positive and negative, by and and for each word in the sentence dataset, calculate the probability of the word which is given in positive/negative sentence, which follows this equation:

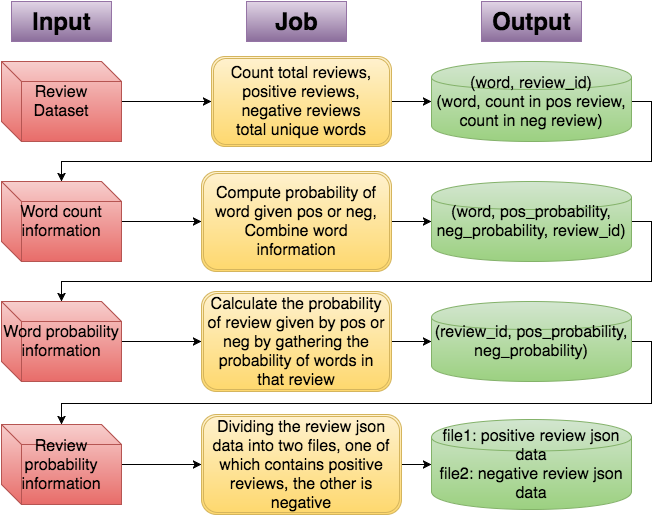
or , where n is the number of words in positive/negative sentences, Vocabulary is the number of unique words occurs in the sentences, w is the number of times the word occurs in the positive/negative reviews. The “1” on the numerator is to avoid error when w is zero. We also can assign very small numbers to w if the “word” doesn’t occurs in some situation.

Finally, after assigning every single word in the sentence dataset a positive possibility and a negative possibility, we can combine those in order to compute the possibility of a new sentence given positive/negative like so:

,

Consequently we can compare those two probabilities, and , if >, then we can say such sentence is more likely to be positive, and vice versa.

Here is the explanation of MapReduce Implementation of Naïve Bayes Algorithm and the process shown as the following figure:



*Mapreduce Flow Chart(4 steps of Naive Bayes text Classifier)*

There are four MapReduce jobs in for text classification. First, the review dataset is the original data source as input for the first job. In this first job, we count the total number of review, including positive reviews, negative reviews, and all the unique words from all the review documents. We represent the output as key-value pairs of word and the ids of review where the words are present. What is more, the number of times that the word occurs in that all reviews will be output with corresponding word as key-value pairs as well. Taking the first job output as input, the second job compute each word probability given positive and negative, and combine the corresponding review ids with probability as key-value output. In third one, each probability of review given positive outcome and negative outcome is calculated by outputting the review id as key, and each probability of words that appear in the reviews as value in the map function, so we can gather all the word probabilities that corresponds to a one specific review and multiply all of them with the positive probability. The product of the multiplication would be the probability of review given positive or negative such that we can determine that the review is positive if its positive probability is bigger than negative one, otherwise the review can be regarded as negative. In order to divide the whole review dataset into a positive json file and a negative json file, in the final MapReduce job, we take the output of the last job and the original review dataset as input. Therefore, we can retrieve that json data of positive review from the original review dataset with the review id corresponding to the positive outcome, and then put all the positive review data into one independent file. And it is the same way to output the negative review data to another file as well.

# Web User Interface Integration

We use jQuery, Bootstrap for our web interface and python on our server.

## Websocket

Getting data from hive is very slow and using join query to get data triggers MapReduce job, which usually take a while.

Http connection only can send one response for each request. If we use http connection, user has to wait a pretty long time for data. Web socket connection can send data to client even if it didn’t get request.

We slice join query to several simple queries, build a search index, use websocket to exchange data between server and browser and store all data from server in LocalStorage of browser to make the response faster.

Each simple query is much faster than join query. We use web socket to send data one by one. For client, to get all data will still take a long time, but very fast get first data.

## LocalStorage

We store all data from server into LocalStorage of browser, user can sort, filter existing data without send and get data from server. We use this technique in our category change function. We also use reverse index for categories mapping.

# Conclusion and improvements

For a three weeks project, our team tried our best to implement everything we have learnt in class into practice, especially the concepts of Big-Data way of dealing with problem, and implementation of MapReduce. Moreover, we tried to integrate all the aspects of our project into a whole product, in our case, a web user interface. By going through our project architecture, we also faced a lot of problems so this means there are also spaces for us to improve our project. From the result perspective, the review classification is not very accurate, sometimes there are obvious mistake in the output of positive and negative classification. We think it's because the review dataset is not positive-negative balanced. Furthermore, we have limited background of statistics and probability, so when we want to study the user’s rating habit, it’s very difficult to find an efficient and reasonable way of doing that. When using our web interface to get the data through websocket, sometimes the responding speed is slow, we think it is because we choosed hive as our data warehouse, the query and join behind scene actually run mapreduce jobs when we retrieve data. So we may consider more options and optimize the performance as well. In the end of our project, we think we should have used Github platform at the first place for co-work efficiency and version control.

# Reference

<https://en.wikipedia.org/wiki/WebSocket>

https://en.wikipedia.org/wiki/Naive\_Bayes\_classifier