Helping Current and Potential Businesses by Providing Business Analytics Using Yelp Dataset

Big Data Miners - Ravi Bisla, Immad Imtiaz, Shariful Islam

1 Problem Definition and Challenges

According to Bloomberg, 8 out of 10 business fail within the first 18 months. A whopping 80% crash and burn. For any new business that needs to open on a place it is necessary to find you future competitors (other similar business in that area) and strategize based on the performance of those business. In this project we build a web-based business analytics solution, that helps new business strategize using Yelp dataset. Our web based analytics answers the following question for any new business.

1. **What are the similar businesses (competitors) around the area we want to open our new business?** Helps to decide whether the place we have chosen for our new business is good or not or whether the vicinity of this place is already saturated with similar businesses.
2. **How these future competitors are performing?** This helps to analyze whether the nature of business we want to open would be successful at the location we have chosen or not.
3. **What is the expected customer traffic on each day of peak and each hour of day?** This helps the new business to manage staff and other resources based on the expected customer traffic so that they can utilize these resources in an optimal way.
4. **Areas where other similar businesses are doing good and areas where these businesses are lacking?** This helps the new business benchmark their quality and service accordingly. The new business can do better on the areas where all these businesses are lacking and capitalize. They can also know their minimum quality and service benchmarks based on where these similar businesses are doing well.
5. **What the users are thinking?** What the users are thinking about a business is very important. A business may learn about user reactions, what they like and dislike and use that knowledge to make informed decision. We try to provide a solution for this as well.

2 Big Data Solution

We address the problems stated above using big data technology. We describe the solution in the following steps.

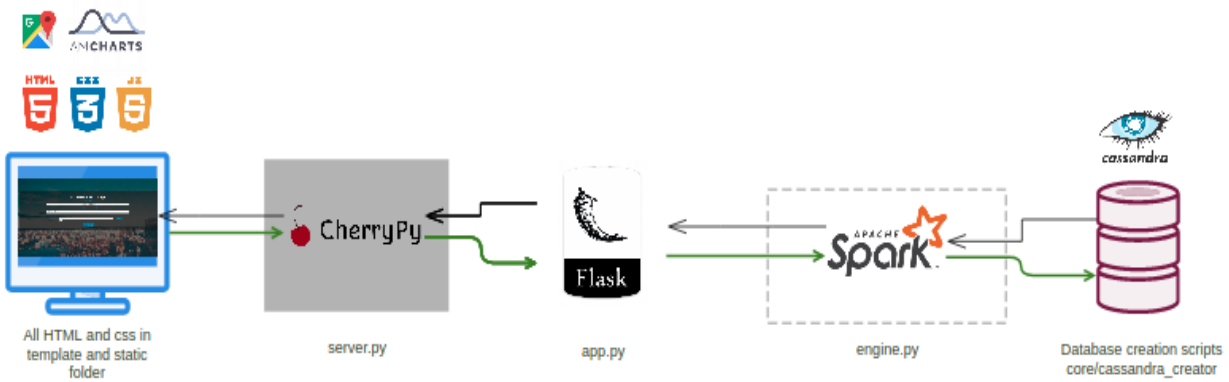
2.1 Dataset Used

Yelp has opened a subset of its huge data for personal, educational, and academic purposes [1]. The dataset we are using is 5.8GB and it has business information of **12 metropolitan areas**. We download the the *.json format dataset. The dataset has six *.json files. Few of the files are not relevant to this project. The files that are relevant to our project are:

- **business.json:** (156,000 businesses): This contains business-specific information such as business ID, business name, address, state, latitude, longitude, review counts, overall rating for all the unique business. We use this file to get information about business categories, location, ratings etc.
- **checkin.json:** This file contains all the available check-in in information. We use this file to get information about check-in.
- **tip.json:** This file contains 1,000,000 tips by 1,100,000 users. We use this file to find out all the text tips given by users. We use the tip for sentiment analysis.
- **review.json:** It contains 4,700,000 reviews. We can use the reviews for sentiment analysis as well.

2.2 Application architecture and tools and technologies used

The technologies we used are shown in the diagram below. We used Spark dataframes and RDD for data transformation our engine.py that does all run-time calculation uses spark. We use **Flask** to create our web application. The precomputed data is on **Cassandra**. We use **Google Location API**'s, **amcharts**, **HTML**, **CSS** and **JavaScript** on front-end. The arrows in the diagram below represent the flow of request-response.



Complete Web Application Architecture

2.3 Data transformation and pre-processing

All the data transformation and preprocessing of data are done in **yelp_spark** package. The pre-processed data is stored in relevant Cassandra tables. The creation script of Cassandra Keyspace and tables can be found in **core/cassandra_creator.py**.

- Transformed the business list attributes and stored them as individual business categories for each business_id.
- For the sentiment analysis we used a dataset [2] of around 8000 words which is divided between positive and negative words. The positive words are given a weight of +1 and the negative words are given a weight of -1. We download that dataset and and put it in the Cassandra database (stored in Cassandra tables **positive_words** and **negative_words**).

2.3.1 Check-in Data Analysis

The check-in data is in the nested JSON format, we transformed this data in a way so that we can store check-in count for each day.

- We were not able to import the JSON by reading it directly as a dataframe.
- We formed the nested dictionaries so that we can return the tuples (business_id, day, hour, checkin_count) and then later saved them in Cassandra tables.

```
for day, value in passed['time'].items():
    for hour, checkin in passed['time'][day].items():
        yield passed['business_id'], day, hour, checkin
```

2.3.2 Sentiment Analysis

In the Yelp dataset we have two files, namely, **review.json** and **tip.json**. In the files we have review and tips (like short review) from different users against different businesses. Since, in our application we aim to provide an idea about the competitor businesses, our objective is to show the set of words that are positive and negative. We follow the following steps to find out the positive and negative words from user reviews.

Step1: We read the information of tip.json file from the cassandra database and we read the reviews provided by the user. We use a function named text_process(), which takes the review text, removes all the punctuations first. Then we use a list of stopwords from nltk.corpus and remove those words from the text and convert the text to a list of words. Now we can convert review and tip to a dataframe who has business_id and review and tips as a list of words.

Step 2: Then we want to find all the tips that are associated with a single business. Therefore, we create a dataframe with a single column which contains all the business_ids. Now we commit a left outer join on business dataframe with review dataframe on business_id. So, after the join operation we have multiple rows for a single business_id.

Step 3: Now we want to aggregate all the list of the words against a single entry of business_id. We add all the list of words into a single list considering the business_id as a key value using reduce by key technique.

At the end of this step we have a dataframe with business_id in one column and all the words in the reviews as a list in a column name ‘word’.

Step 4: In this step we want to separate the positive and negative words. This is the vital part. One way of doing this to find the sentiments associated with all the words present in the review text list. However, finding sentiments associated with words need significant machine learning and enough data to learn sentiment of the words. Moreover, we do not have any ground truth to evaluate that model. So we follow a simpler approach. We use a database of words (introduced above) which is divided between positive and negative words. The positive words are given a weight of +1 and the negative words are given a weight of -1. Now we use a function named **explode** from pyspark.sql.functions, which converts the dataframe with business_id and list of words into a dataframe with business_id and single word at each row.

Now if we commit an inner join of this exploded dataframe with the dataframe of the positive words on ‘word’, we will find all the business with positive words. Similarly, we can find a dataframe with business_id and negative words. We can now use the groupBy property of dataframe and collect_list from pyspark.sql.functions, we can again make a list of all the positive words and negative words. Now we have a dataframe with columns business_id, positive words and negative words.

Step 5: From the dataset, we see that, all the businesses do not get a review or tip. Thus, we need to join the dataframe from the previous step with the initial dataframe with all the business_ids. After a left outer join on business_ids we find a dataframe which has all the business_ids with list of positive and negative words list, when available. Now we write this data to Cassandra database (in table **business_sentiments**).

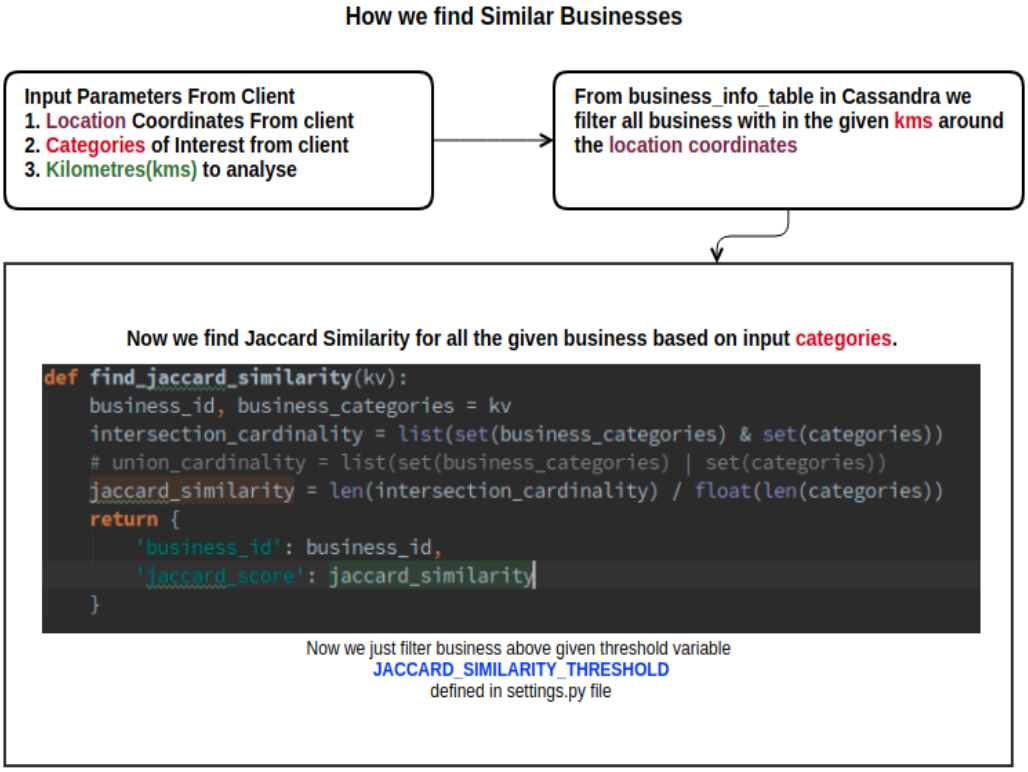
2.4 Data Processing on the run-time (engine.py)

1. To find locations that are within a certain radius distance of a given latitude/longitude, we used the SELECT statement based on the Haversine formula. A snapshot of the code is given below:

```
BUSINESS_RADIUS_SQL = """
    SELECT latitude, longitude, city, business_id, name, stars, review_count,
    (6371 * acos(cos(radians(%s)) * cos(radians(latitude)) * cos(radians(longitude)
    - radians(%s)) + sin(radians(%s)) * sin(radians(latitude )))) AS distance
    FROM %s
    having distance < %s ORDER BY distance
"""
```

Here's the SQL statement that will find the closest locations that are within a radius of given distance passed from front end. It calculates the distance based on the latitude/longitude of that row and the target latitude/longitude, and then asks for only rows where the distance value is less than passed distance, orders the whole query by distance. We used 6371 to convert into kilometers.

2. First, we find similar business, the process can be explained in the diagram below.

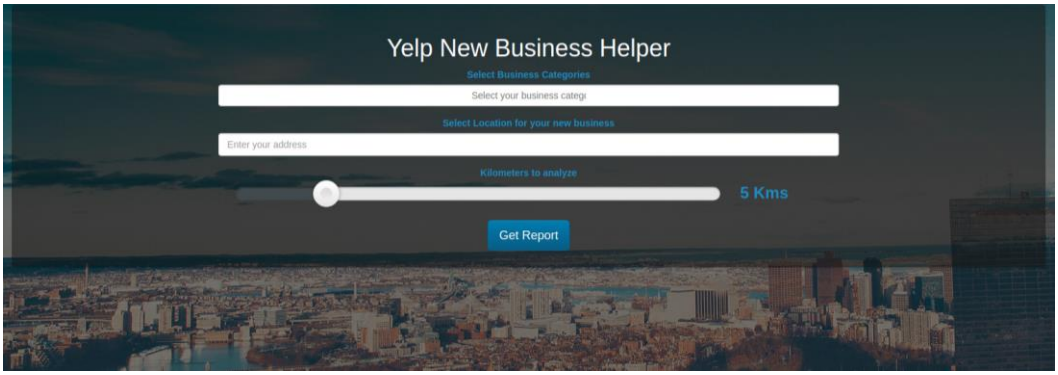


3. Next based on the similar business we found we find the results from our Cassandra database that we pre-calculated.
2. We find top 5 competitors from these businesses based on distance from our given location, reviews and ratings.
3. We find the average check ins by day and hour for these businesses using our precomputed data in Cassandra **(**tables are **business_time_check_in** and **business_day_check_in)**
4. Next, we find all positive and negative keywords for all similar business in **business_sentiments** table in Cassandra that were pre-calculated.
5. Next, we just send the data to front-end for report creation.

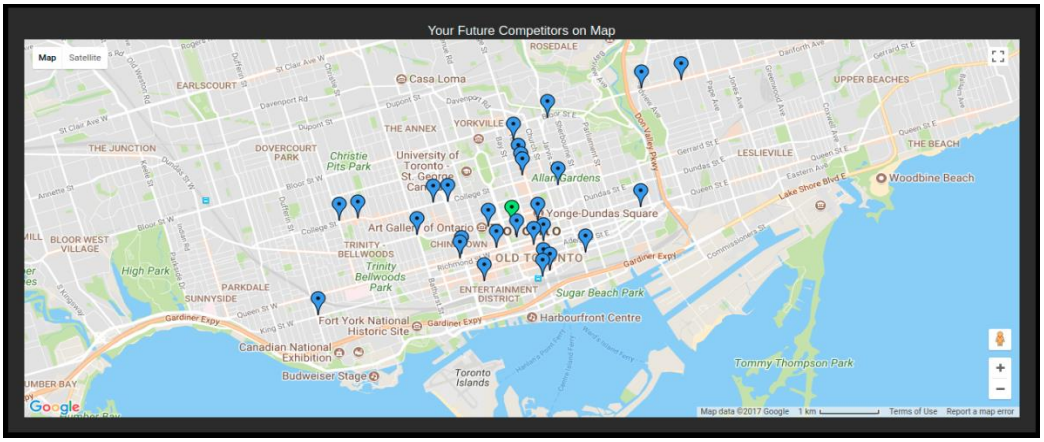
3. Results

Our web application provides solution to problems defined for any new business.

1. We have a web front-end where we can put query. The following image shows our front-end.



2. Top five **similar businesses (future competitors)** with same categories as chose from the web client enables new business to find all the competitors around the given address in km radius provided. Helping them to make right decision in choosing the place. The map below shows that this place has a lot of competition for this new business in green.

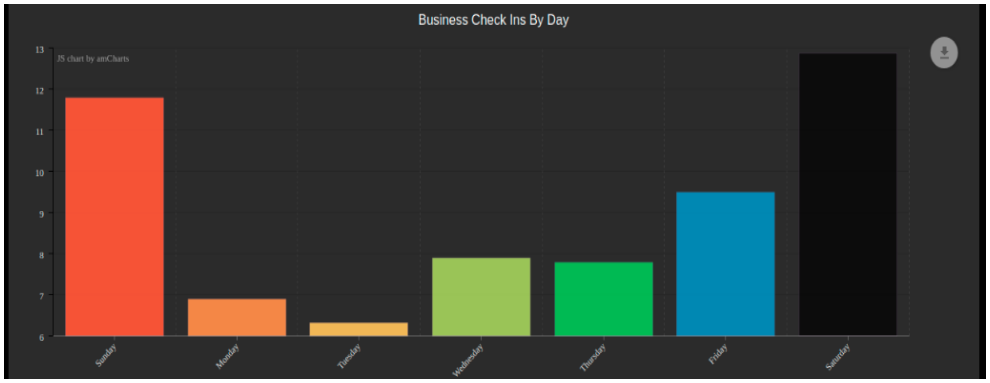


3. **Top five possible competitors** help to strategies business strategies studying these businesses. They are extracted based on distance from the location, reviews and ratings.

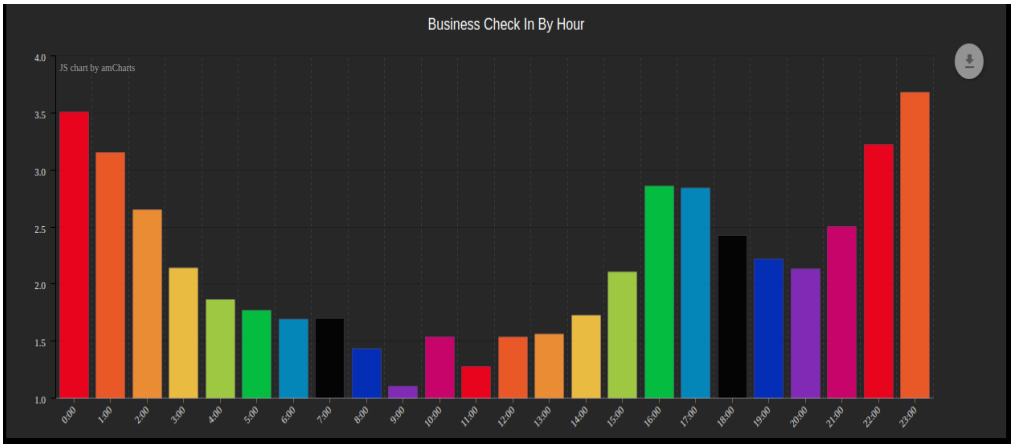
Top Competitors

Business Name	Distance (kms)	Reviews	Ratings
Patty and Franks	3.89	18	4.5
Hidden Burger	1.08	30	4.5
Capitano Burgers & Gelato	1.52	79	4.0
Mr Tasty Fries	0.27	20	4.0
Harvey's Restaurants	1.05	17	3.5

4. **Average day wise check-ins** of similar business help the business analyze how the inflow of the customers would be each day for the nature of his business on given location. This would help either to make a decision in choosing location or strategise on resource planning such as staff or part time employees. **(Checkins is correlated with customers in flow)**



5. **Hourly average check-ins** help the new business choose the most optimal business hours of operation and allocated resources efficiently such as more number of waiters at the hours when check ins are more.



6. Provided positive and keywords from the tips of similar business that helps the new business understand and develop their quality benchmarks.

4. New technologies learned

1. **amCharts.js** We learned amcharts to develop the bar charts on the web.
2. **Bokeh** We initially generated graphs on the backend using this python library and rendered on the front end but this was removed as generating graph on backend made the application slow.
3. **Flask** Since we were not familiar with any of the python frontend frameworks we learned this framework in order to create our app.

5. References

[1] Yelp Open Dataset - https://www.yelp.com/dataset
[2] Word database with sentiment - https://github.com/daniel-acuna/python_data_science_intro/blob/master/data/sentiments.parquet.zip

6. Source Code

Github URL: https://github.com/immad-imtiaz/Yelp-Recommender-Big-Data
GitLab SFU URL: https://csil-git1.cs.surrey.sfu.ca/iimtiaz/Yelp-Recommender-Big-Data/tree/master

Self Assessment:

- Getting the data: We get data from Yelp challenge [1]. [0 points]
- ETL: We copy the data to cluster, then move it to HDFS. Then we processed the data depending on our need. Put the preprocessed data in Cassandra database. [2 points]
- Problem: Help new businesses using Yelp data. [2 points]
- Algorithmic work: Finding similarity using Jaccard similarity, writing function for finding businesses within a radius using latitude and longitude, finding average check-in information, finding positive and negative words from user reviews using sentiment analysis. [6 points]
- Bigness/parallelization: The size of the data is 5.8 Gigabyte, which we pre-processed using Spark and put the data in Cassandra. Our backend is also running Spark. As we are doing everything using big data tools, our solution is scalable, and we have a fairly large amount of dataset. [3 points]
- UI: User interface to the results, possibly including web or data exploration frontends. [2 points]
- Visualization: Visualization of analysis results. [2 point]
- Technologies: [3 point]