Streaming and Static Recommendation System of Businesses

Gopi Chand Vanka, Srinivas Lingamgunta, Abhitej Boorla, Ravindra Daliparthy, *The University of Texas at Dallas*

***Abstract*—Technology has become a part and parcel of our lives. Reviews are playing a pivotal role for users in decision making. People are sharing their experiences on products or services in the form of reviews, which enables other users for their decision making. It's hard for the user to go through each and every review, and it might mislead the user to use a wrong service or a product, if he/she goes by the review given by a user with opposite taste. So we came up with a system which recommends the user depending on his needs.**

***Index Terms*—Static Recommendation System, Streaming Recommendation System, Apache Kafka, Apache Zookeeper, Apache Spark**

I. Introduction

THESE days reviews play vital role in every individual's life for choosing the best product or service. They have enhanced the probability of buying the right product, going to a good movie, eating at the best restaurant etc. But sometimes they could mislead a user in making a bad choice if he/she has opposite tastes when compared to the user who has written the review. So to overcome this problem we have built a system, which helps the user in choosing the right restaurant according to his/her requirements.

Our work comprises of two recommendation systems in which, one uses the Static data and other uses the real time streaming data. Both of these systems are discussed in the later sections of this paper.

To withstand the high competition, restaurants try to improve their quality to satisfy their customers. We processed tips data on daily basis through streaming and recommended top restaurants to the user.

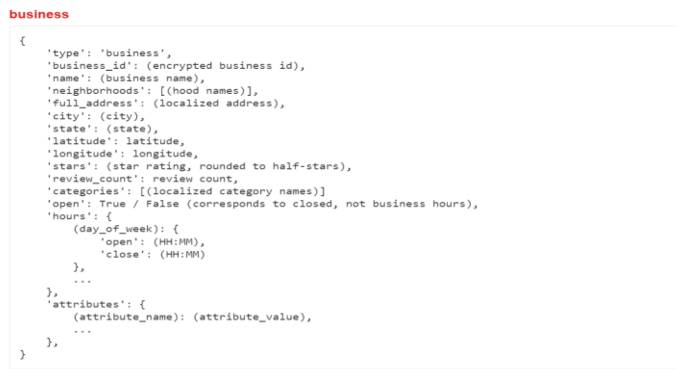
Apache Zookeeper, Apache Kafka servers are used to stream our static data, which are discussed in the later sections of this paper.

Our systems were experimented on Yelp dataset and the results, future works are also discussed in the later sections. Java, Scala, R and Python were used for accomplishing the project.

A. Yelp Dataset

Yelp hosted Academic dataset challenge, which has been used as our dataset, which comprises of business, review, tip, checkin, user information in JSON format.

Business data comprises of City, State, Latitude, Longitude, Business Id, Name, Stars etc. Detailed description and JSON format of the business dataset is shown in Fig. 1[1].

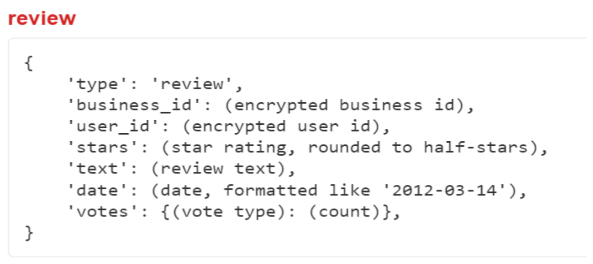


business

```
{
    '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),
        ...
    },
}
```

Fig. 1. Business Dataset

Review data comprises of User_Id, Review_Id, Stars, Date, Text etc. Detailed description and JSON format of the Review dataset is shown in Fig. 2[1].

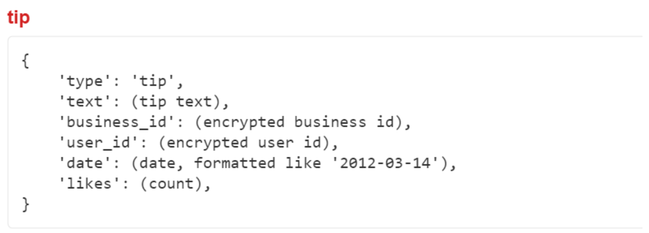


review

```
{
    '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)},
}
```

Fig. 2. Review Dataset

Tip dataset comprises of Text, User_Id, Business_Id. Detailed description and JSON format of the Tip dataset is shown in Fig. 3[1].



tip

```
{
    'type': 'tip',
    'text': (tip text),
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'date': (date, formatted like '2012-03-14'),
    'likes': (count),
}
```

Fig. 3. Tip Dataset

The data comprises of various businesses reviews and we have contained ourselves to Restaurants domain. For the convenience of analyzing we have converted the JSON format to CSV files using Python script.

II. Background

A. Apache-Spark [2]

Apache Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

The availability of RDDs facilitates the implementation of both iterative algorithms, that visit their dataset multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications (compared to Apache Hadoop, a popular MapReduce implementation) may be reduced by several orders of magnitude. Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.

Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.

B. Spark Streaming [2]

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, thus facilitating easy implementation of lambda architecture. However, this convenience comes with the penalty of latency equal to the mini-batch duration. Other streaming data engines that process event by event rather than in mini-batches include Storm and the streaming component of Flink. Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.

C. Apache Kafka [3]

Apache Kafka is an open-source stream processing platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Its storage layer is essentially a "massively scalable pub/sub message queue architected as a distributed transaction log," making it highly valuable for enterprise infrastructures to process streaming data. Additionally, Kafka connects to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library.

D. Apache Zookeeper [4]

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

E. MLlib Machine Learning Library [2]

Spark MLlib is a distributed machine learning framework on top of Spark Core that, due in large part to the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by Apache Mahout (according to benchmarks done by the MLlib developers against the Alternating Least Squares (ALS) implementations, and before Mahout itself gained a Spark interface), and scales better than Vowpal Wabbit. Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which simplifies large scale machine learning pipelines, including:

- summary statistics, correlations, stratified sampling, hypothesis testing, random data generation
- classification and regression: support vector machines, logistic regression, linear regression, decision trees, naive Bayes classification
- collaborative filtering techniques including alternating least squares (ALS)
- cluster analysis methods including k-means, and Latent Dirichlet Allocation (LDA)
- dimensionality reduction techniques such as singular value decomposition (SVD), and principal component analysis (PCA)
- feature extraction and transformation functions
- optimization algorithms such as stochastic gradient descent, limited-memory BFGS (L-BFGS)

III. Static Recommendation System

For this system, we planned to recommend the user, a restaurant based on his/her location and preferred food type. For improved results we planned to build the model only if user has rated at least one of the nearby restaurants

Step 1: Depending on the location of the user, list of nearby restaurants was obtained.
Step 2: Restaurant list is filtered based on the cuisine choice.
Step 3: If user has already rated any of the restaurants in the list from the Step 2, then we built an ALS model using the current user and all users' recent ratings on the list of the restaurants obtained in the Step 2.
Step 4: Predicted the user rating on the list of restaurants obtained.
Step 5: Recommended the top 5 predicted restaurants.
Step 6: If the user hasn't rated any of the restaurants then we have recommended the top 5 rated restaurants nearby.

Steps are shown below:

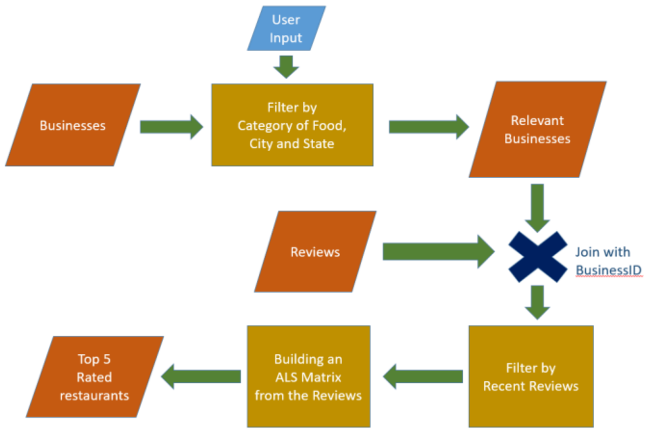


Fig. 3. Static Recommendation System flow chart

System is built on Apache Spark using Scala and R is used for pre-processing and extracting the data, results are discussed in later sections.

IV. Streaming Recommendation System

For coping up with the cut throat competition restaurants work on negative reviews for getting better. For example, if the restaurant which didn't serve better pizza yesterday tries to improve. We wanted to recommend the user depending on the reviews given on the same day, which helps him/her to choose the best restaurant on that day. For our project, we couldn't get the real time streaming data to work on, so we came up with our own Java program using Kafka and Zookeeper servers which streams the monthly tips data of the restaurants nearby.

Step 1: Depending on the location of the user, list of nearby restaurants was obtained.
Step 2: Took the information from the user, depending on the food he/she wants to eat Example: pizza.
Step 3: Streamed data using Java program with count 1000, formed bigrams on tips and found the positive bigrams like "good pizza" and collected the frequency of the positive bigrams.
Step 4: Recommended the restaurants with high positive bigram frequency.

Steps are shown in the below flow chart:

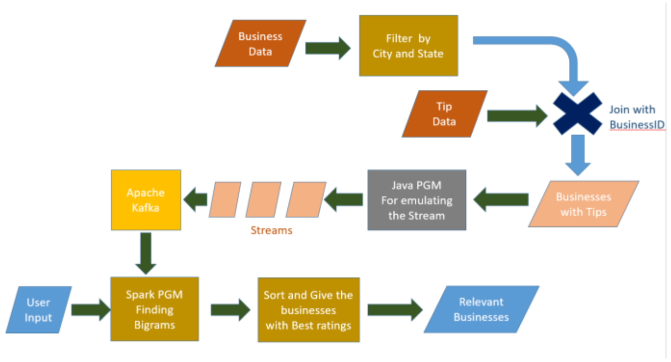
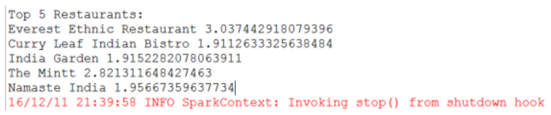


Fig. 4. Streaming Recommendation System flow chart

System is built on Apache Spark using Scala, Apache Kafka and Apache Zookeeper Servers and R is used for pre-processing and extracting the data, results are discussed in later sections.

V. Results

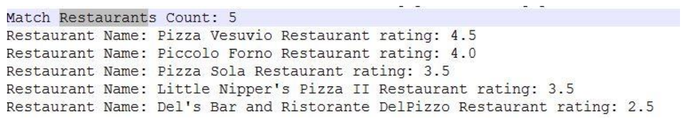
Results of the Static Recommendation System is displayed below which consists of top 5 restaurants obtained through ALS model. Restaurant along with the predicted rating is shown.



```
Top 5 Restaurants:
Everest Ethnic Restaurant 3.037442918079396
Curry Leaf Indian Bistro 1.9112633325638484
India Garden 1.9152282078063911
The Mintt 2.821311648427463
Namaste India 1.95667359637734
16/12/11 21:39:58 INFO SparkContext: Invoking stop() from shutdown hook
```

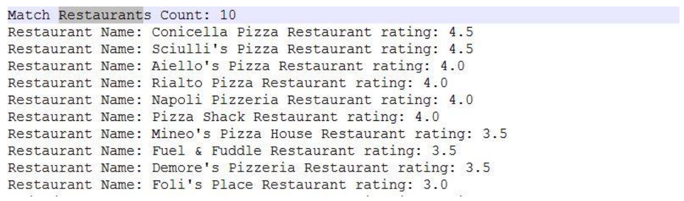
Fig. 5. Static Model output1

Results of the Streaming Recommendation System is shown in the below figures, we can see that output has changed each time the data is streamed, which means that the user is being recommended different restaurants for each new data stream. Which says that daily reviews on a particular restaurant changes and the user will be recommended the best restaurants based on that day review.



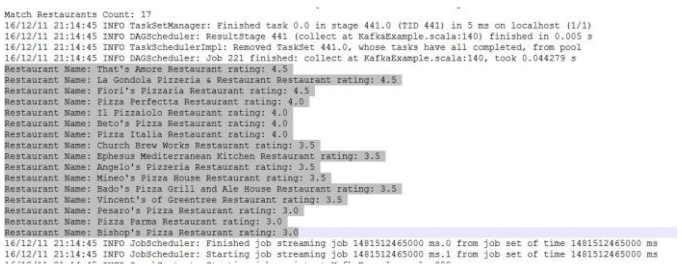
```
Match Restaurants Count: 5
Restaurant Name: Pizza Vesuvio Restaurant rating: 4.5
Restaurant Name: Piccolo Forno Restaurant rating: 4.0
Restaurant Name: Pizza Sola Restaurant rating: 3.5
Restaurant Name: Little Nipper's Pizza II Restaurant rating: 3.5
Restaurant Name: Del's Bar and Ristorante DelPizzo Restaurant rating: 2.5
```

Fig. 6. Streaming Model output1



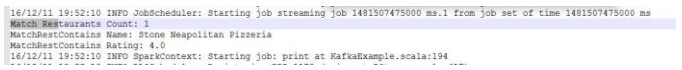
```
Match Restaurants Count: 10
Restaurant Name: Conicella Pizza Restaurant rating: 4.5
Restaurant Name: Sciulli's Pizza Restaurant rating: 4.5
Restaurant Name: Aiello's Pizza Restaurant rating: 4.0
Restaurant Name: Rialto Pizza Restaurant rating: 4.0
Restaurant Name: Napoli Pizzeria Restaurant rating: 4.0
Restaurant Name: Pizza Shack Restaurant rating: 4.0
Restaurant Name: Mineo's Pizza House Restaurant rating: 3.5
Restaurant Name: Fuel & Fuddle Restaurant rating: 3.5
Restaurant Name: Demore's Pizzeria Restaurant rating: 3.5
Restaurant Name: Foli's Place Restaurant rating: 3.0
```

Fig. 7. Streaming Model output2



```
Match Restaurants Count: 17
16/12/11 21:14:45 INFO TaskSetManager: Finished task 0.0 in stage 441.0 (TID 441) in 5 ms on localhost (1/1)
16/12/11 21:14:45 INFO DAGScheduler: ResultStage 441 (collect at KafkaExample.scala:140) finished in 0.005 s
16/12/11 21:14:45 INFO TaskSchedulerImpl: Removed TaskSet 441.0, whose tasks have all completed, from pool
16/12/11 21:14:45 INFO DAGScheduler: Job 221 finished: collect at KafkaExample.scala:140, took 0.044279 s
Restaurant Name: That's Amore Restaurant rating: 4.5
Restaurant Name: La Gondola Pizzeria & Restaurant Restaurant rating: 4.5
Restaurant Name: Fiori's Pizzaria Restaurant rating: 4.5
Restaurant Name: Pizza Perfectta Restaurant rating: 4.0
Restaurant Name: Il Pizzaiolo Restaurant rating: 4.0
Restaurant Name: Beto's Pizza Restaurant rating: 4.0
Restaurant Name: Pizza Italia Restaurant rating: 4.0
Restaurant Name: Church Brew Works Restaurant rating: 3.5
Restaurant Name: Ephesus Mediterranean Kitchen Restaurant rating: 3.5
Restaurant Name: Angelo's Pizzeria Restaurant rating: 3.5
Restaurant Name: Mineo's Pizza House Restaurant rating: 3.5
Restaurant Name: Bado's Pizza Grill and Ale House Restaurant rating: 3.5
Restaurant Name: Vincent's of Greentree Restaurant rating: 3.5
Restaurant Name: Pesaro's Pizza Restaurant rating: 3.0
Restaurant Name: Pizza Parma Restaurant rating: 3.0
Restaurant Name: Bishop's Pizza Restaurant rating: 3.0
16/12/11 21:14:45 INFO JobScheduler: Finished job streaming job 1481512465000 ms.0 from job set of time 1481512465000 ms
16/12/11 21:14:45 INFO JobScheduler: Starting job streaming job 1481512465000 ms.1 from job set of time 1481512465000 ms
```

Fig. 8. Streaming Model output3



```
16/12/11 19:52:10 INFO JobScheduler: Starting job streaming job 1481507475000 ms.1 from job set of time 1481507475000 ms
Match Restaurants Count: 1
MatchRestContains Name: Stone Neapolitan Pizzeria
MatchRestContains Rating: 4.0
16/12/11 19:52:10 INFO SparkContext: Starting job: print at KafkaExample.scala:194
```

Fig. 9. Streaming Model output4

VI. Future Work

For Streaming Model, we could not get enough data for each day, so we experimented it by streaming 1000 records in each stream, lack of data can be solved if restaurants were made to collect the reviews from every customer. We have worked only on positive reviews, if we were able to perform sentimental analysis on the tips data, then results can be enhanced.

VII. Conclusion

Our work proposed a new approach to recommend user better restaurants based on analyzing daily reviews. This helps the user to choose the better restaurant on the same day. As discussed in the future work section, if we are able to extend work by implementing sentimental analysis we can acquire much better results.

References

[1] https://www.yelp.com/dataset_challenge
[2] https://en.wikipedia.org/wiki/Apache_Spark
[3] https://en.wikipedia.org/wiki/Apache_Kafka
[4] https://zookeeper.apache.org/
[5] http://nverma-tech-blog.blogspot.com/2015/10/apache-kafka-quick-start-on-windows.html