Rashid Khwaja

Sunwoo Lee

Yong Li

Wenxin Mo

VoramatePasharawipas

Language trends by location in Yelp Reviews

I. INTRODUCTION

For this project, we aim to train the LSTM model by each region. The regions are five in total, including Canada, Europe, US South, US Southwest, US Midwest. The purpose of training LSTM is to generate “real fake review” that will bypass the Yelp’ auto fake review detecting system. General summary of our process is the following. First, we utilize Apache Spark MapReduce technique to divide the Yelp review data by each region. Then five subset review data are used to train five different LSTM model representing each region. Then to extract useful information from review data, we again use Spark to get average length of review and the top most used words in reviews by each rating and region. Using the information from average length and top common words, we can replicate the real reviews’ characteristics and thus generate “real fake reviews”.

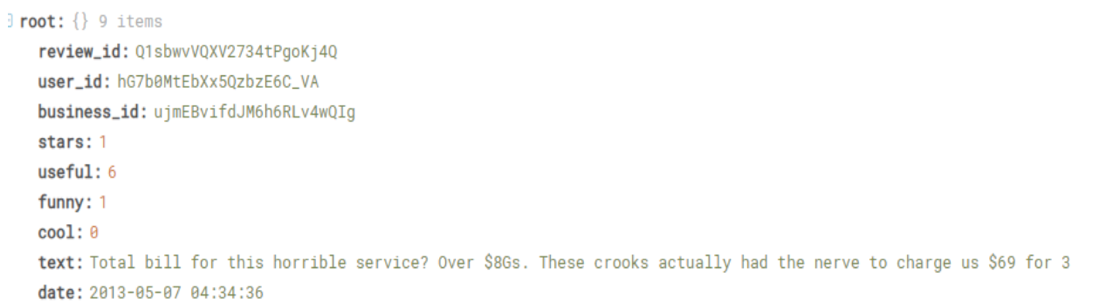
The details of our process will be further described below which includes our PySpark program design of average length and common words from reviews of different rating for each region. Also, the design of LSTM model will be depicted as well.

II. STEP ONE

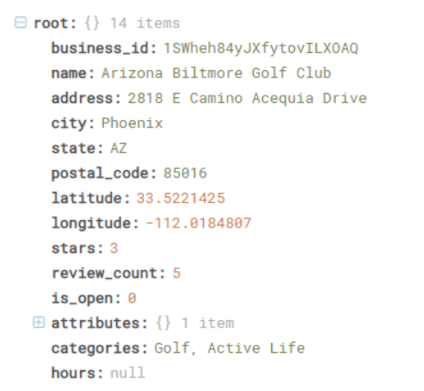
*A. DATA PREPARRATION*

1.Read yelp\_business.csv and yelp\_review.csv files to Spark Data Frame

Here is a sample data of yelp\_review.csv



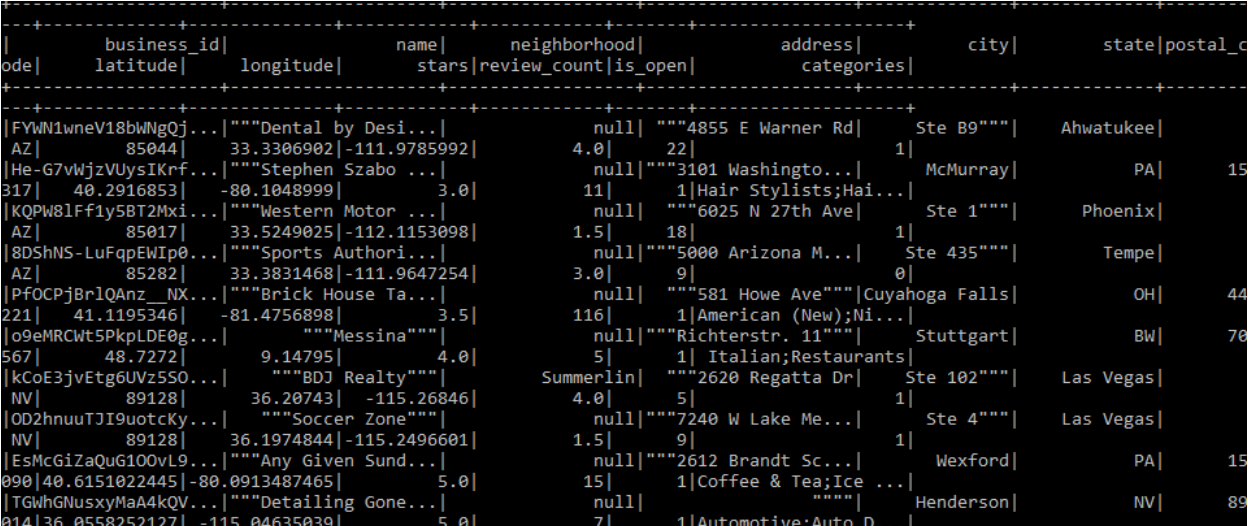
Sample data of yelp\_business.csv



Yelp\_review spark dataframe

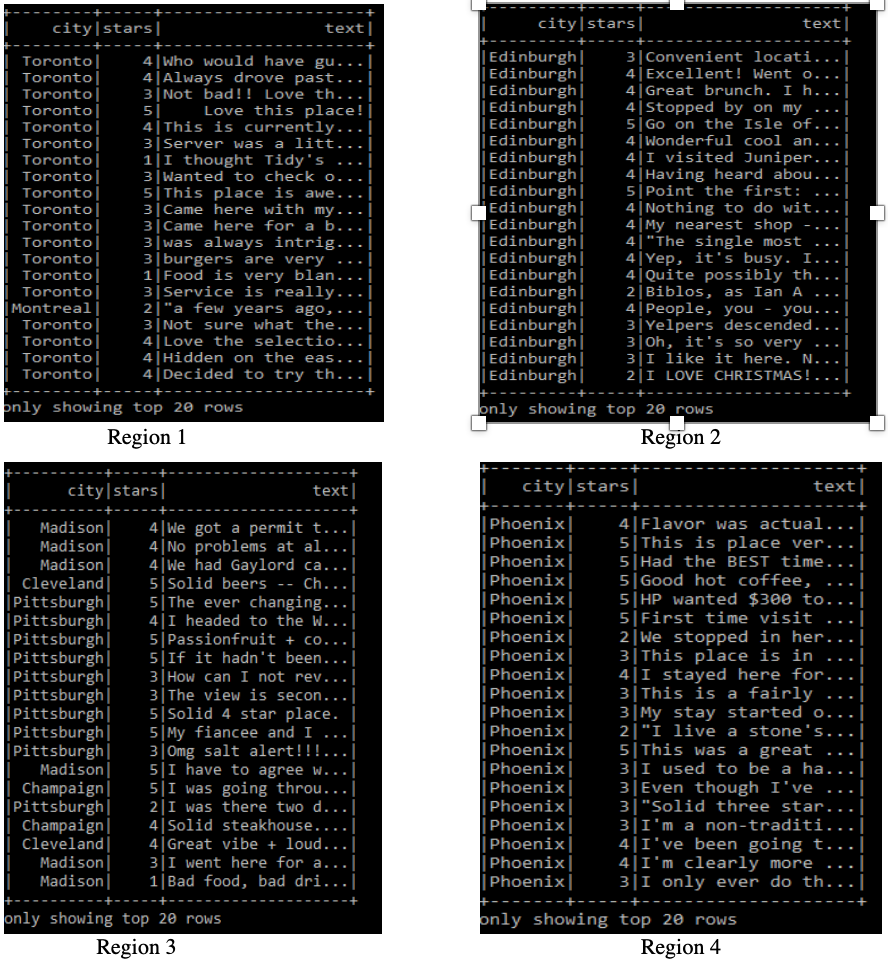


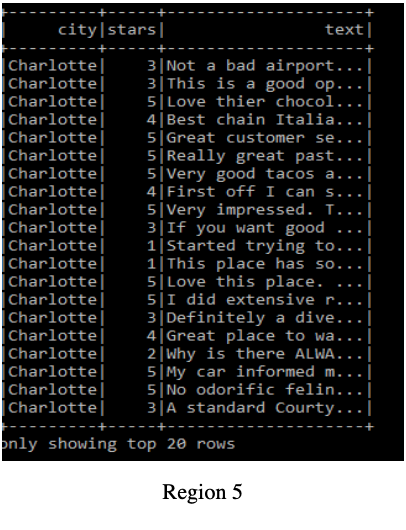
Yelp\_business spark dataframe



1. Join these two datasets so that we can get reviews base on region. We also need to select only columns that we need and remove missing values. The way we join these two datasets, we used Spark DataFrame’s join function (inner join on business\_id)
2. Split the dataset by region Using Spark SQL.

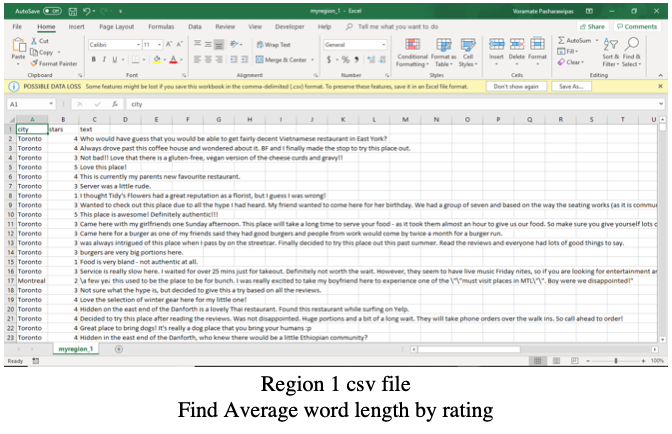
Below are the results after the split using Spark SQL





3. Save to csv file in case we need to do further transformation separately by region.

“region\_1.coalesce(1)” command will aggregate all the result from every nodes into 1 master node and then write out as 1 csv file.



II. STEP TWO

*A. Create a new RDD from the external file*

First, using the coding method to create SparkContext to connect local and the spark. Then, create a new rdd from the external file named "region\_3.csv." Moreover, using split and len function for each comment lines to get rating and length for each review from region\_3. Then remove all punctuation. (Example from region\_3)

*B. The idea of Getting the average length of comments*

The big idea is to divide the total amount of comments words by a total number of rating. For example, we need to know the total number of words in one-star rating comments, and the number of one-star rating comments. Then use the total number of words divided by the total amount of a one-star rating.

Therefore, I use mapValues to create key-value pairs.



After that, I need to aggregate the same star rating’s total amount. I use a reduceBykey function to get the result. (reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1])). The idea is that when we in the same key( ‘1.00’; ‘2.00’; ‘3.00’ and so on), their value should have (x,y) format. Moreover, x[0]+y[0],x[1]+y[1] means add the first value add second value together in the same rating, the following count the amount of this key. Then we have:



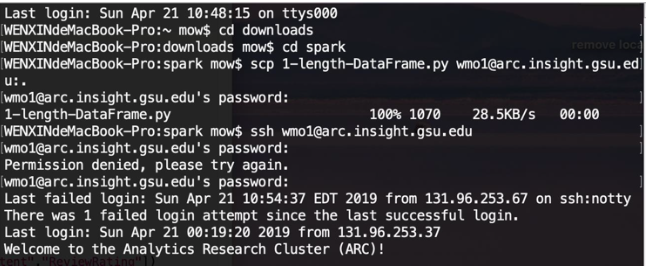
Finally, we can compute the average by mapValues(lambda x:x[0]/x[1]) function. Which is use the total amount of words divided by the total number of the comments in the same rating.

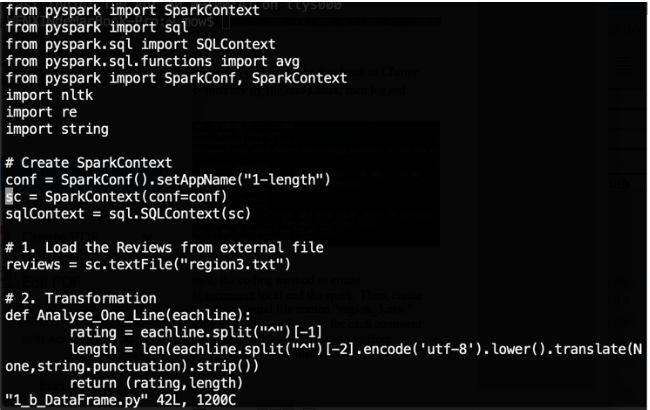
The result can be shown as followed:



*C. Create DataFrame to Calculate the Avg.length in Cluster*

First I copied my py.file into Linux, then logged into a cluster;





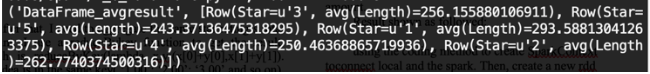
And I removed the local master and split the data into two columns by "^," they are "star" and "text

."

To have text content in word by word format, I split the data again.

Then I convert them as DataFrame; after that, I used the groupBy function in Rating to get the average content amount.

The result is shown as followed:



The result is no difference between data frame method and non-data frame method.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Word per review | Characters per review | Average word length |
| Canada | 51 | 264 | 5.176470588 |
| Europe | 47 | 259 | 5.510638298 |
| US-Midwest | 59.5 | 309 | 5.193277311 |
| US-Southwest | 63 | 320 | 5.079365079 |
| US-South | 61 | 315 | 5.163934426 |

Words per rating level

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Canada | Europe | US-Midwest | US- Southwest | US-South |
| 1-Star | 63.1 | 55.7 | 73.5 | 82.6 | 79.7 |
| 2-Star | 73.8 | 50.6 | 66.8 | 71.2 | 71.1 |
| 3-Star | 37.2 | 49.4 | 61.8 | 62.7 | 61.3 |
| 4-Star | 46.2 | 47.6 | 58.2 | 57.1 | 56.6 |
| 5-Star | 31.1 | 45.4 | 56.1 | 57.1 | 55.7 |

III. STEP THREE

In this Step, we need to figure out the top 20 words for each rating comment.

*A. Create a new RDD from the external file*

In the first step, similar to Step one. We need to get the rdd file in (rating, comments words) format. So, I used the split function for each comment lines to gain rating and every word for each region\_3 reviews. However, different from Step one, I need to remove the stopwords before getting the frequency of words.

This is the words that after removed stopwords and some common popular but useless to our LSTM model words.

*B. The idea of Getting top 20 common words*

The big idea is to get the frequency of words in each rating comments. Specifically, after getting the pure dataset, we can use reduceByKey function to get a word count for a particular rating, and then use map function to have the list of word and word count in different rating ((rating, [(word, word count)]), the next step is to use reduceByKey function again to get the list of total word count in a unique rating. Then sorted them to get top 20 common words.



This graph showing that our value reduces by key (rating, word); in other words, it gives us a phrase counting by rating.

By using the map function, we can have a list of the word in different rating. (Example for one-star rating)



*C. Result*

Now I need to sort the frequency of each word for every rating in sorted function. The result can be shown as followed:

1 star rating : [u'back', u'never', u'even', u'us', u'dont', u'order', u'got', u'went', u'told', u'ordered', u'minutes', u'said', u'didnt', u'ever', u'could', u'came', u'restaurant', u'experience', u'worst', u'people']

2 star rating : [u'really', u'ordered', u'back', u'dont', u'got', u'restaurant', u'went', u'even', u'us', u'ive', u'didnt', u'order', u'im', u'better', u'came', u'first', u'much', u'two', u'pretty', u'also']

3 star rating : [u'really', u'pretty', u'bar', u'little', u'restaurant', u'ordered', u'back', u'got', u'ive', u'also', u'dont', u'went', u'im', u'menu', u'much', u'night', u'came', u'didnt', u'first', u'try']

4 star rating : [u'really', u'also', u'back', u'ive', u'little', u'pittsburgh', u'always', u'love', u'well', u'restaurant', u'bar', u'got', u'menu', u'pretty', u'im', u'friendly', u'definitely', u'try', u'first', u'went']

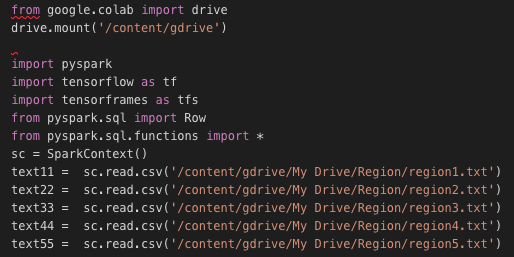
5 star rating : [u'love', u'pittsburgh', u'ive', u'really', u'back', u'always', u'amazing', u'friendly', u'also', u'staff', u'well', u'definitely', u'got', u'restaurant', u'im', u'recommend', u'favorite', u'try', u'even', u'first']

*D. Run LSTM model to predict fake reviews*

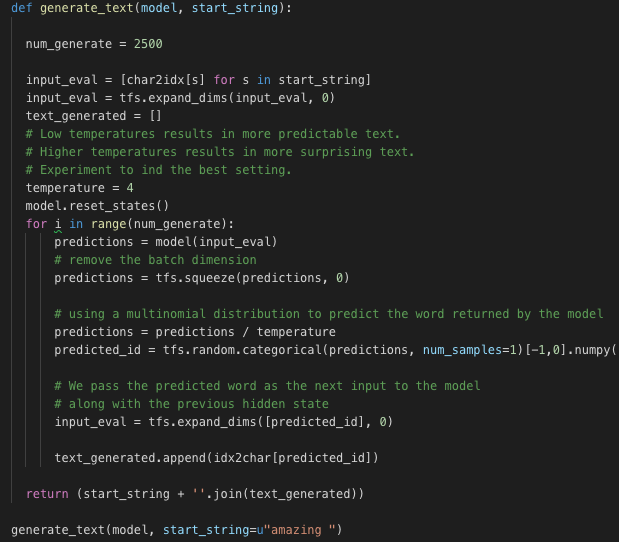
Lastly, we built a Long term short term memory model in pyspark and tensorflow using google colab. First, we had to change the environment path in google colab for the cloud service to be able to run pyspark.



Then we tensorframes to build our LSTM. Tensorframes is a package in spark that lets you build deep learning models on tensorflow from a spark dataframe. First, I imported the necessary packages and loaded the review text as a spark dataframe



Then I used tensorflow to train an LSTM model on the reviews from the spark dataframe. Our model consists of an embedded layer, and LSTM layer, and then a hidden layer. With our trained model, I then made a generator that can predict characters with a given input string.



Then we selected the input string and length according to the average length and top 20 common words, we need to pick some common words to generate fake reviews from our fitted LSTM model.

Here is the result we got for region\_3 with temperature = 1:

Star1: never

My absolute favorite restaurants. We wanted to ...

Too sitting and my new favorite dining experience...

This Blue for the worst restaurants in the buff...

I love the atmosphere-one Star here. Has a local meeting...

Star2: better

Finally made it without has, the be change change

Love the food, we wanted to come up on a Weire...

Another job on my only this I tant often but drach i... Extremely clean, the delite - atmospher'

Star3: really

Really enjoyed the gyro-ope...

This is one of the best pizza in Pittsburgh! We dec...

Makagho is unsustential, it specials, the inforwar... Best meal here various includes quickly back in pret...

Thanks has a pretty good burbars and the service...

Star4: friendly

Finally have 2 beers to eat and checking into the Ba... friendly service tacos and rating workingous and the shakes are my birthdard,…

The only time I could try Murrys this literally...

Star5: first

Love this place!! The brew was good!

Extremely clean, the delite - atmospher'

The only ran has food things about their cigaries i... This restaurant has moving the best buh I...

Lastly, we combined a few predictions so that the word length of our fake review coincides with the average length of that rating within a specific region. After that we connected our computed to a server in each region using NordVPN. The cities we connected to were

Montreal for the Canadian region, Chicago for the US-Midwest region, SF for the US-West region, and Atlanta for the US-South region. We then created a Yelp account and posted our LSTM generated reviews. Below is an example of a posted fake review.

