INFO7250 Engineering Big-Data Systems - Fall 2016

Yelp: Predicting User Ratings for New Business

Section - 01

Sarthak Agarwal

Professor – Dino Konstantopoulos

December 12, 2016

I. Table of Contents

Chapter 1

1. Introduction

1.1 Idea

The idea is to build a predictive system that predicts the degree of likeness of new food joints for users based on users past historical data on a global geographical scale.

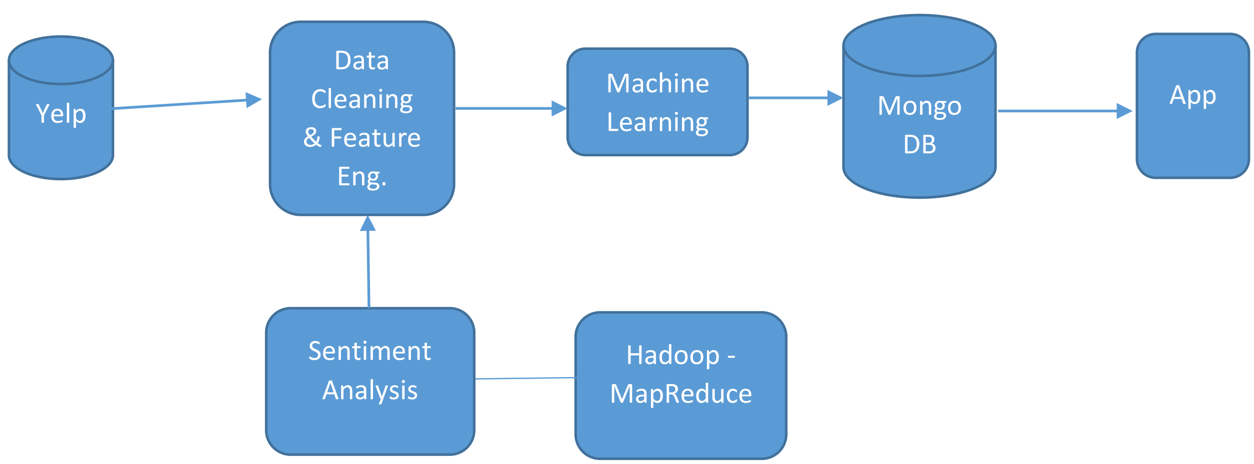
e.g. If user travels to California from Boston, the system will predict which restaurants he will like in his five-mile radius and how will he rate the restaurant based on his/her previous pattern on Yelp.

1.2 Challenge

Out of more than 100 features across five different data feeds, choose the appropriate features or create a new feature to get a high accuracy for each user.

1.3 Approach

A high level diagram/flow of the project:



Chapter 2

2. Dataset Description

The Yelp dataset consists of five data feeds but we primarily work on the below mentioned two feeds.

- Business – Information of all the businesses in Yelp.
- Review – Reviews of all the business.

The data is provided by Yelp on their website which consists of *85,539* businesses and *2,685,066* reviews. The file sizes were 75MB and 2.2GB respectively.

The data is provided as JSON files and their structure is as follows:

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)},
}
```

Chapter 3

3. Data Cleaning & Feature Engineering

3.1 Dataset Preparation

3.1.1 Convert JSON to CSV We used Python to convert JSON to CSV and the code is as follows

```python
# -*- coding: utf-8 -*-

import argparse
import collections
import csv
import simplejson as json

def read_and_write_file(json_file_path, csv_file_path, column_names):
    """Read in the json dataset file and write it out to a csv file, given the column names."""
    with open(csv_file_path, 'wb+') as fout:
        csv_file = csv.writer(fout)
        csv_file.writerow([column_names,])
        with open(json_file_path) as fin:
            for line in fin:
                line_contents = json.loads(line)
                line_contents = line_contents.replace("\n","")
                csv_file.writerow(get_row(line_contents, column_names))

def get_superset_of_column_names_from_file(json_file_path):
    """Read in the json dataset file and return the superset of column names."""
    column_names = set()
    with open(json_file_path) as fin:
        for line in fin:
            line_contents = json.loads(line)
            column_names.update(
                    set(get_column_names(line_contents).keys())
                    )
    return column_names

def get_column_names(line_contents, parent_key=''):

    column_names = []

    for k, v in line_contents.items():
        column_name = "{0}.{1}".format(parent_key, k) if parent_key else k
        if isinstance(v, collections.MutableMapping):
            column_names.extend(
                    get_column_names(v, column_name).items()
                    )
        else:
            column_names.append((column_name, v))
    return dict(column_names)

def get_nested_value(d, key):

    if '.' not in key:
        if key not in d:
            return None
        return d[key]
    base_key, sub_key = key.split('.', 1)
    if base_key not in d:
        return None
    sub_dict = d[base_key]
    return get_nested_value(sub_dict, sub_key)

def get_row(line_contents, column_names):
    """Return a csv compatible row given column names and a dict."""
    row = []
    for column_name in column_names:
        line_value = get_nested_value(
                        line_contents,
                        column_name,
                        )
        if isinstance(line_value, unicode):
            row.append('{0}'.format(line_value.encode('utf-8')))
        elif line_value is not None:
            row.append('{0}'.format(line_value))
        else:
            row.append('')
    return row

if __name__ == '__main__':
    """Convert a yelp dataset file from json to csv."""

    parser = argparse.ArgumentParser(
            description='Convert Yelp Dataset Challenge data from JSON format to CSV.',
            )

    parser.add_argument(
            'json_file',
            type=str,
            help='The json file to convert.',
            )

    args = parser.parse_args()

    json_file = args.json_file
    csv_file = '{name}.csv'.format(name=json_file.split('.json')[0])
```

3.1.2 Used R to analyze the data and prepare it for Machine Learning input.
Following steps were performed:

- Filter the data related to food and similar categories.
- Filter the data for the users that have reviewed more than hundred businesses.
- Extract the relevant features out of 95 attributes in the Business dataset which will be used in our Machine learning algorithm.
- Rename the attributes to more meaningful names.
- Build and structure the data for our input to the Machine Learning Algorithm

Load the Business, Review and User CSVs into dataframe

```
#### BUSINESS
business <- read.csv("D:/GRAD_SCHOOL/Fall2016/Project_Yelp/DatasetsInCSV/DatasetsInCSV/yelp_academic_dataset_business.csv")
head(business)
colnames(business)
```

```
####### REVIEW
file.path <- "D:/GRAD_SCHOOL/Fall2016/Project_Yelp/DatasetsInCSV/DatasetsInCSV/yelp_academic_dataset_review.csv"
library(data.table)
review <- fread(file.path)
```

```
##### USER
user <- read.csv("D:/GRAD_SCHOOL/Fall2016/Project_Yelp/DatasetsInCSV/DatasetsInCSV/yelp_academic_dataset_user.csv")
head(user)
str(user)
```

Filter the data related to food and similar categories

```
business_foodData <- sqldf('select distinct * from business where categories like \'%Food%\'
                            or  categories  like \'%Restaurants%\'
                            or categories  like \'%Lounges%\'
                            or categories like \'%Nightlife%\'
                            or categories like \'%Bars%\'')
str(business_foodData)
```

Filter the data for the users that have reviewed more than hundred businesses

```
xx <- sqldf('select r.user_id, count(r.business_id)
            from review_user_business r, business_foodData b
            where r.business_id = b.business_id
            group by r.user_id
            having count(r.business_id) > 100')
head(xx)

xx2 <- sqldf('select a.user_id, b.business_id from
             xx a,
             business_foodData b,
             review_user_business r
             where a.user_id = r.user_id
             and b.business_id = r.business_id')
head(xx2)

write.csv(xx2,'D:/GRAD_SCHOOL/Fall2016/Project_Yelp/DatasetsInCSV/DatasetsInCSV2/Our_Users_GreaterThan100_Businesses.csv')
```

Rename the attributes to more meaningful names.

```
nrow(business_foodData_relevantFeatures)

colnames(business_foodData)[32] <- "attributes_Price_Range"
colnames(business_foodData)[40] <- "attributes_Accepts_Credit_Cards"
colnames(business_foodData)[44] <- "attributes_Take_out"
colnames(business_foodData)[55] <- "attributes_Delivery"
colnames(business_foodData)[60] <- "attributes_Wheelchair_Accessible"
colnames(business_foodData)[42] <- "attributes_Good_For_lunch"
colnames(business_foodData)[68] <- "attributes_Good_For_dinner"
colnames(business_foodData)[20] <- "attributes_Good_For_brunch"
colnames(business_foodData)[33] <- "attributes_Good_For_breakfast"
colnames(business_foodData)[51] <- "attributes_Takes_Reservations"

nrow(business_foodData_relevantFeatures)
```

Build and structure the data for our input to the Machine Learning Algorithm.

We structured the data based on the input that our ML algorithm needs. The input to the algorithm will be something like this:

Users_Id, Business_id, ----Relevant Features---, Label

```
AllusersAllBusinessAllFeatures <-
  sqldf('select r.user_id, r.business_id, a.latitude, a.longitude, a.review_count,
        a.attributes_Price_Range, a.attributes_Accepts_Credit_Cards, a.attributes_Take_out,
        a.attributes_Delivery, a.attributes_Wheelchair_Accessible, a.attributes_Good_For_lunch,
        a.attributes_Good_For_dinner, a.attributes_Good_For_brunch, a.attributes_Good_For_breakfast,
        a.attributes_Takes_Reservations, round(a.stars) as business_stars, a.attributes_Parking_lot,
        a.attributes_Parking_street, a.attributes_Parking_garage, a.attributes_Parking_valet,
        a.attributes_Parking_validated, a.attributes_Good_For_Groups, a.attributes_Good_For_kids,
        a.attributes_Ambience_casual, a.attributes_Ambience_romantic, a.attributes_Ambience_upscale,
        a.attributes_Ambience_classy,
        r.review_avg_stars
        from business_foodData a, review_final_round r
        where a.business_id = r.business_id')

colnames(AllusersAllBusinessAllFeatures)
nrow(review_final_round)
```

3.2 Feature Generation using Sentiment Analysis

We generated a new feature using sentiment analysis of Reviews in Yelp dataset. This feature is a sentiment score of a business based on the sentiments of all the reviews that the business has reviewed. According to us this feature is a true representation of what user thinks about the business.

We wrote the code in R and tried performing sentiment analysis, but since the review file is approximately 2.2 GB and we have to process each word of each review, R was going out of memory. Hence we implemented it using Hadoop Map Reduce and ran the code in Distributed mode.

Our MapReduce algorithm is as follows:

1. The input to our Mapper is the reviews file in JSON format and used Google's GSON library to read the file and assign it to the class.
2. In the Mapper we calculate the sentiment score of all the reviews and output the Business_id as key and a combination of Review_id and sentiment score as value.

```
public class SentimentMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        Gson gson = new Gson();
        // used gson object so that we can assign the headers of the json into a class.

        if (!value.toString().contains("\"text\"")) {
            return; // Not a review we're interested in.
        }

        // takes the values from the json and stores in the Class so that we can use the reviews as a String in our program.
        Review review = gson.fromJson(value.toString(), Review.class);

        DoubleWritable sentimentValue = new DoubleWritable(review.calculateSentimentValue());
        Text businessId = new Text(review.business_id);

        // making the value as review_id+ sentimentValue (which is calculated by using lexicon.ttf file which stores the sentiment score of that particular word.
        Text value_text =  new Text(review.review_id + ":" + sentimentValue);

        context.write(businessId, value_text);
    }
```

3. In the Reducer, we split the value and extracts the sentiment score and calculate the average sentiment score for each business.
4. The output of the reducer is the Business_id and its sentiment score.

```java
public class RatingReducer extends Reducer<Text, Text, Text, DoubleWritable> {

    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        double total = 0;
        double length = 0;

        List<Double> dd = new ArrayList<Double>();

        // key=business_id and values= review_id + sentiment_score
        // split the value on : so that we can get the sentiment score of the against that business id

        for (Text t : values ){
            String[] arrVal = t.toString().split(":");
            String sentiment_score = arrVal[1];
            Double sentiment_score_double = Double.parseDouble(sentiment_score);
            dd.add(sentiment_score_double);
        }

        // store the sentiment value in list of double so that we can aggregate while its on the loop.
        for (Double sentiment : dd) {
            total += sentiment;
            length++;
        }

        //calculate the average sentiment score of each business

        double average = total / length;

        if(key.toString().equalsIgnoreCase("--0ZoBTQWQra1FxD4rBWmg"))
            System.out.println("Found it again");

        //emit bbusiness_id, sentiment_score
        context.write(key, new DoubleWritable(average));
    }
```

Since the reviews data was of 1.9 GB, we ran the map reduce job on fully distributed system. It took around 10 minutes to complete the job. Below are the screenshot of the job-tracker sandbox.

JOBTRACKER SAND – Fully Distributed Mode

Cluster Summary (Heap Size is 123 MB/889 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
2	1	4	1	2	2	4.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201612101541_0007	NORMAL	manasi-desktop\manasiladdha	Yelp Sentiment Analysis	11.42%	35	4	3.80%	1	0	NA

JOBTRACKER SAND – Fully Distributed Mode

Cluster Summary (Heap Size is 123 MB/889 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	4	1	2	2	4.00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201612101541_0007	NORMAL	manasi-desktop\manasiladdha	Yelp Sentiment Analysis	100.00%	35	35	100.00%	1	1	NA

Chapter 4

4. Machine Learning

4.1.1 Data Splitting

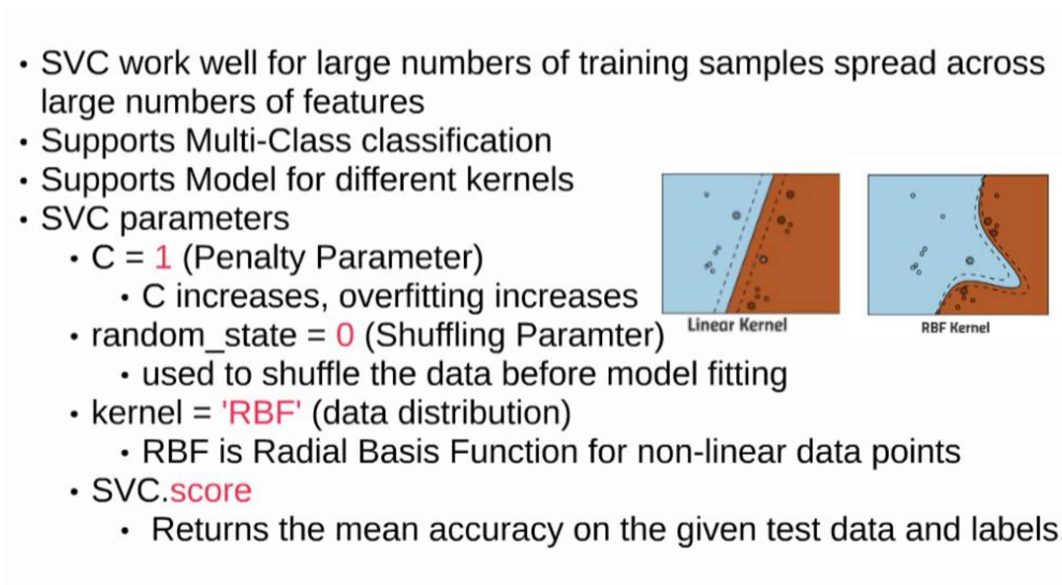
Using python, scikit-learn library, we split the data into training and test data using *train_test_split* function of *sklearn.model_selection* library. We set the function to shuffle the data and then split it into 80 percent training and 20 percent test data.

4.1.2 Algorithm

We implemented the two models using scikit libraries: support vector machines (SVM) and random forest classifiers. These algorithms were implemented with the defaults from scikit, which can be found in their user guide. However, we investigated the types of kernels for SVM and the regularization parameters for SVM.

a. Random Forest:
While implementing Random forest we used the scikit implementation and the results were over fitted. i.e. the difference between the training and testing accuracy was huge.

b. SVC (Support Vector Classification):



Algorithm Overview

We experimented with SVM using scikit's SVM with linear, polynomial and Gaussian kernels. We also performed a parameter sweep and the results were as follows:

Training Accuracy	Testing Accuracy	Parameters
0.623397036	0.536425359	c = 1, kernel = "rbf"
0.714505147	0.433934534	c =2, kernel = "rbf"
0.600952494	0.427557664	c = 1, kernel = "linear"
0.775047514	0.413936467	c =1, kernel = "rbf", no preprocessing
0.617386536	0.426587355	c = 1, kernel = "rbf", randomState = 0
0.577037933	0.423913976	Linear SVC, pentaly = l2
0.545430948	0.413995582	Linear SVC, pentaly = l2, loss = "hinge"

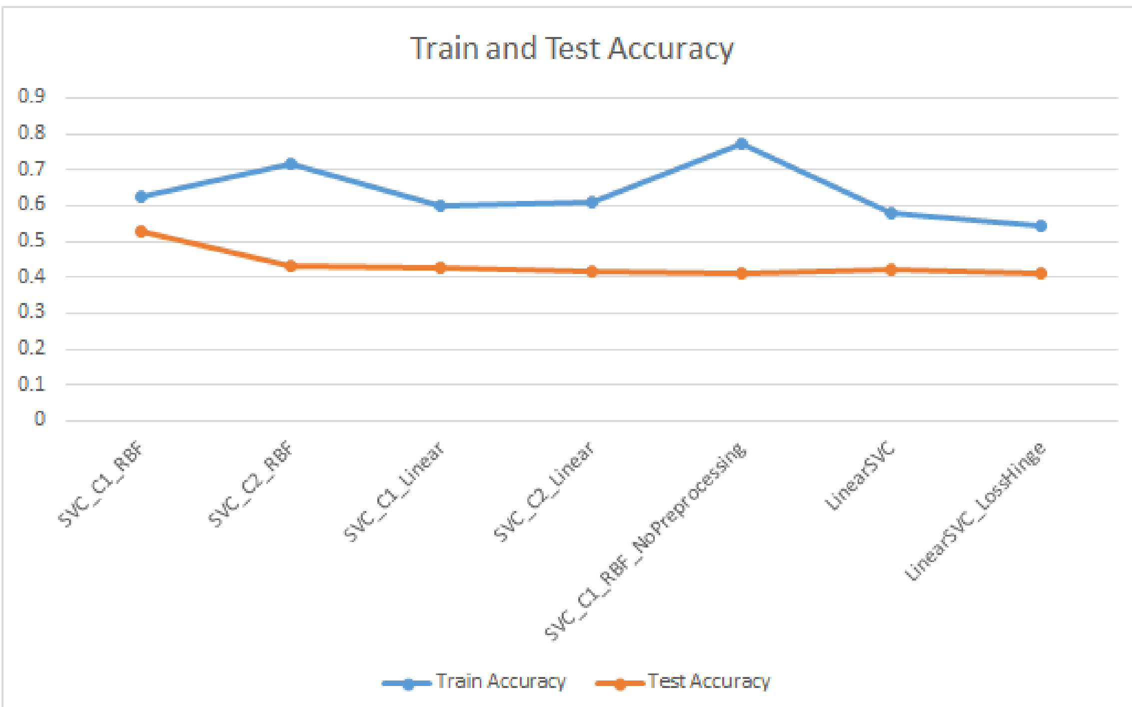


Figure 1: SVM results with different parameters

Based on the above results, we settled on C-SVC with RBF kernel to perform our prediction.

With C-SVC algorithm, our machine learning model acheives a decent accuracy of 62% and we are able to predict the rating of restaurants which have not been reviewed by a specific user nicely.

Radial Basis Function kernel network scale well to large numbers of features in the input space. The pre-processing module further provides a utility class StandardScalar that implements the Transformer API to compute the mean and standard deviation on the training set so as to be later reapply the same transformation on the testing set. The 'c' parameter trades off misclassifying of training examples against simplicity of the decision surface. When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes.

Python Code for Modelling

```
print('------Model Making Started-------')
results = defaultdict(list)
users_models = defaultdict()
for user, business_dict in our_user_our_business_dict.items():
    final_features = []
    final_labels = []
    for business, features_list in business_dict.items():
        final_features.append(features_list[:-1]) #everything except the last item
        final_labels.append(features_list[-1]) #last label

    final_features = numpy.array(final_features)
    final_labels = numpy.array(final_labels)

    X_train, X_test, y_train, y_test = train_test_split(final_features, final_labels,
                                                        test_size=0.2, random_state=0)

    scaler = preprocessing.StandardScaler().fit(X_train)
    X_train_transformed = scaler.transform(X_train)
    clf = SVC(C=1, gamma=0.2).fit(X_train_transformed, y_train)
    print(clf.get_params())
    X_test_transformed = scaler.transform(X_test)
    train_accuracy = clf.score(X_train_transformed, y_train)
    test_accuracy = clf.score(X_test_transformed, y_test)

    if train_accuracy >=0.0 and test_accuracy>=0.0:
        users_models[user] = clf
        results[user].append(X_train.shape)
        results[user].append(X_test.shape)
        results[user].append(train_accuracy)
        results[user].append(test_accuracy)
```

4.1.3 Finding new business

After creating the model for all the users based on the features of their already reviewed businesses, our python code finds out the businesses that the user has not yet reviewed and on which we will perform the prediction.

There were 982 Users and approximately 36991 were non predicted businesses for each users. So the final file with all the features required for the prediction resulted in a 3.47 GB file on which the prediction was done.

Python Code for new businesses identification

```
print("------- Diff the businesses for users --------- ")
all_business = set(all_business)
all_users_new_businesses = defaultdict(list)
for u, business_set in all_users_old_businesses.items():
   diff_business = all_business.symmetric_difference(business_set)
   all_users_new_businesses[u] = list(diff_business)

print("------GENERATING NEW BUSINESSES WITH FEATURES FOR USERS --------")
all_users_new_businesses_features = defaultdict(dict)
final_results =[]
for u, business_list in all_users_new_businesses.items():
   for b in business_list:
       final_results.append([u , b] +  all_business_features[b] )
       all_users_new_businesses_features[u].update({ b : all_business_features[b]})

test_user = list(all_users_new_businesses_features.keys())[0]
print(all_users_new_businesses_features[test_user])

# write results to a csv
resultFile = open("FilteredUsers_NewBusinesses_OurFeatures.csv",'w')
wr = csv.writer(resultFile, delimiter=',', lineterminator='\n' )
wr.writerow(['user_id', 'business_id', 'attributes_Price_Range','attributes_Accepts_Credit_C
              'attributes_Take_out','attributes_Delivery','attributes_Wheelchair_Accessible',
              'attributes_Good_For_lunch','attributes_Good_For_dinner','attributes_Good_For_
              'attributes_Good_For_breakfast','attributes_Takes_Reservations',
              'latitude','longitude','business_stars', 'review_count', 'sentimental_rating' ]
wr.writerows(final_results)
```

4.1.4 Prediction

To predict, we use the SVC-model generated for each user to predict the User Rating for these non-predicted businesses. We use an AWS EC2 instance with 32GB, 1TB configuration and it took approx. 52 minutes 29seconds to complete the prediction.

For example,

user_id	records_trained_on	records_tested_on	train_accuracy	test_accuracy	actual_avg_stars	predicted_avg_stars
1	80	21	0.7375	0.666666667	2.9	3
2	116	30	0.784482759	0.766666667	4.04	4
3	96	24	0.791666667	0.75	4.75	5

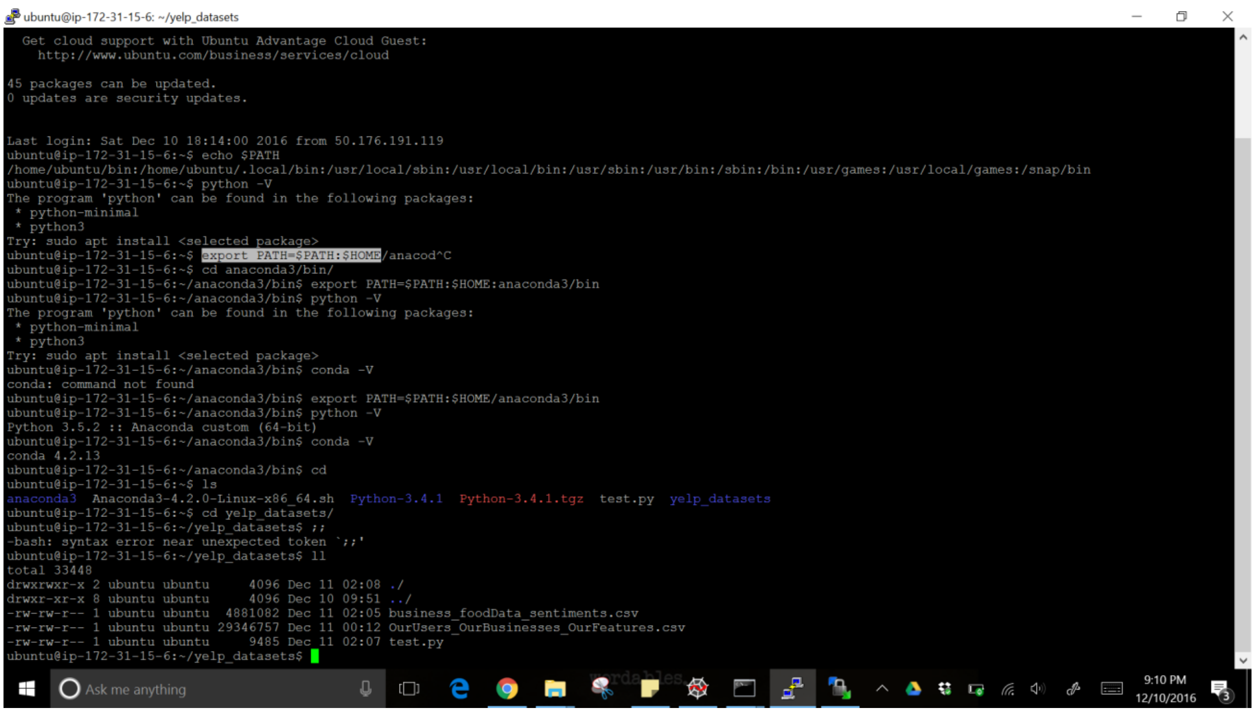
User_id 1 was trained on 80 records and tested on 21 records and we got a train accuracy 73% and a test accuracy 66%. The actual average stars were 2.9 for the user and the model predicted an average stars of 3 for new businesses.

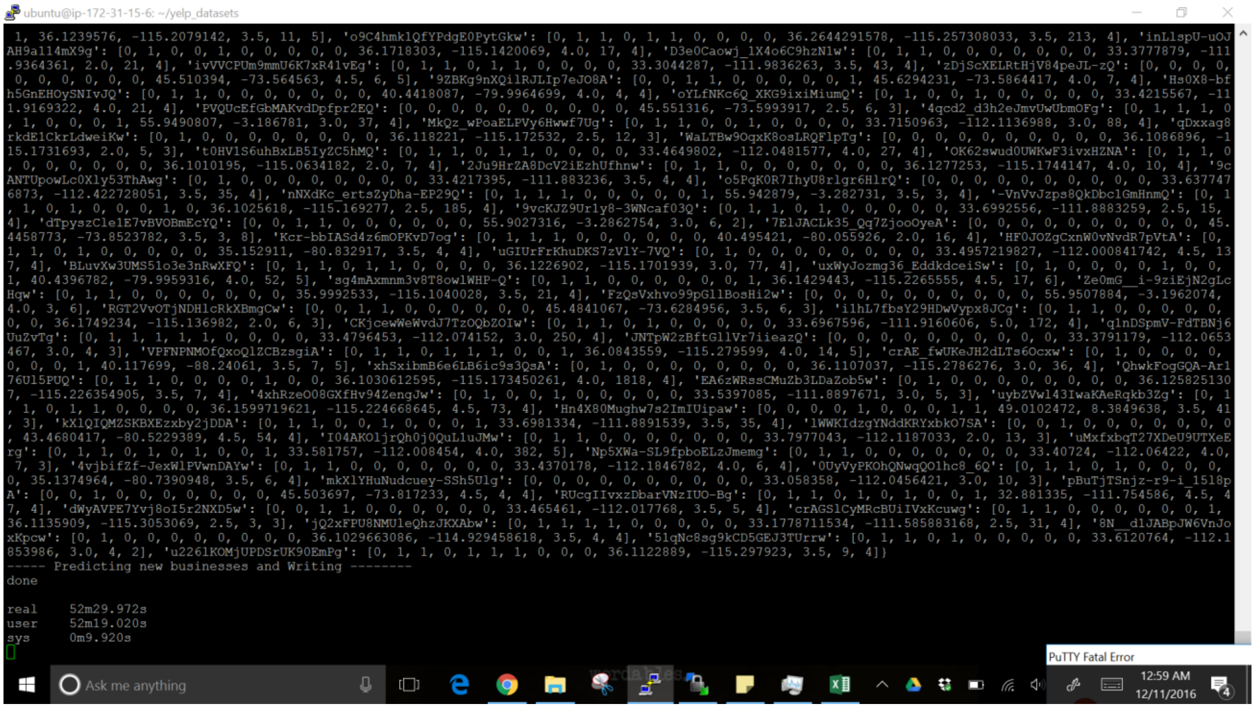
Python Code for Predicting

```
print("----- Predicting new businesses and Writing --------")
resultFile = open("FilteredUsers_NewBusiness_Predictions.csv",'w')
wr = csv.writer(resultFile, delimiter=',', lineterminator='\n' )
wr.writerow(['user_id', 'business_id' , 'prediction'])
all_users_new_businesses_predictions = defaultdict(dict)
for user, business_list in all_users_new_businesses_features.items():
    for new_business, new_business_features in business_list.items():
        predicted_label = users_models[user].predict([new_business_features])
        wr.writerow([user, new_business, predicted_label[0]])
        all_users_new_businesses_predictions[user] = { new_business : predicted_label[0] }

resultFile.close()
print("done")
```

AWS EC2 Instance

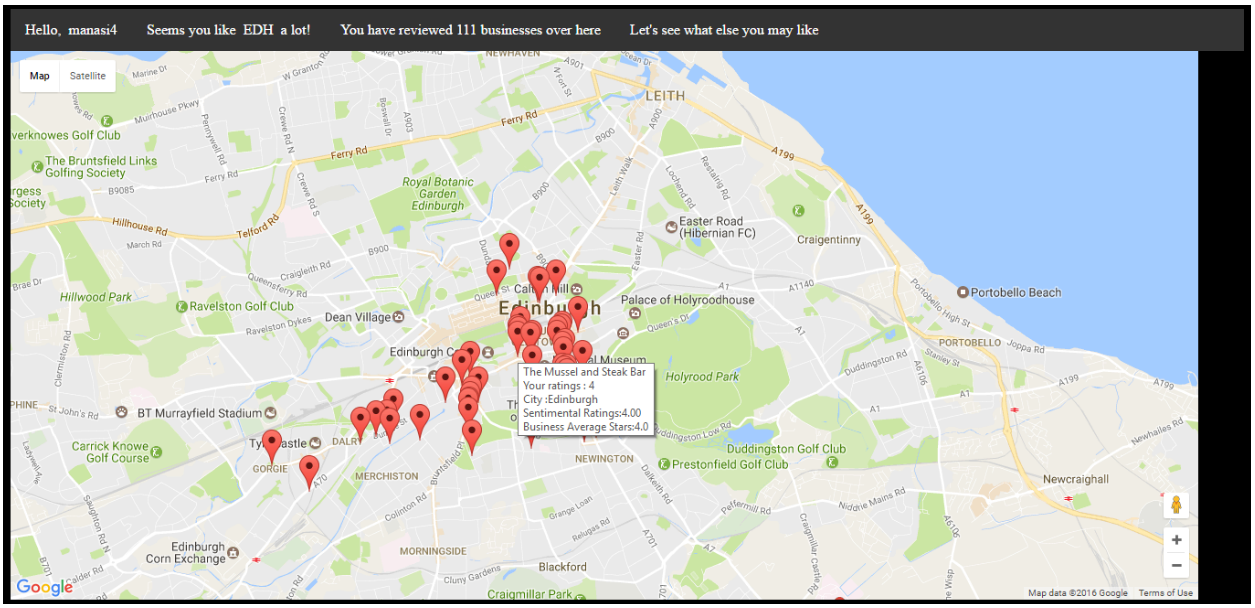



ubuntu@ip-172-31-15-6: ~/yelp_datasets
----- Predicting new businesses and Writing ---------
done
real 52m29.972s
user 52m19.020s
sys 0m9.920s
PuTTY Fatal Error
Ask me anything
12:59 AM
12/11/2016

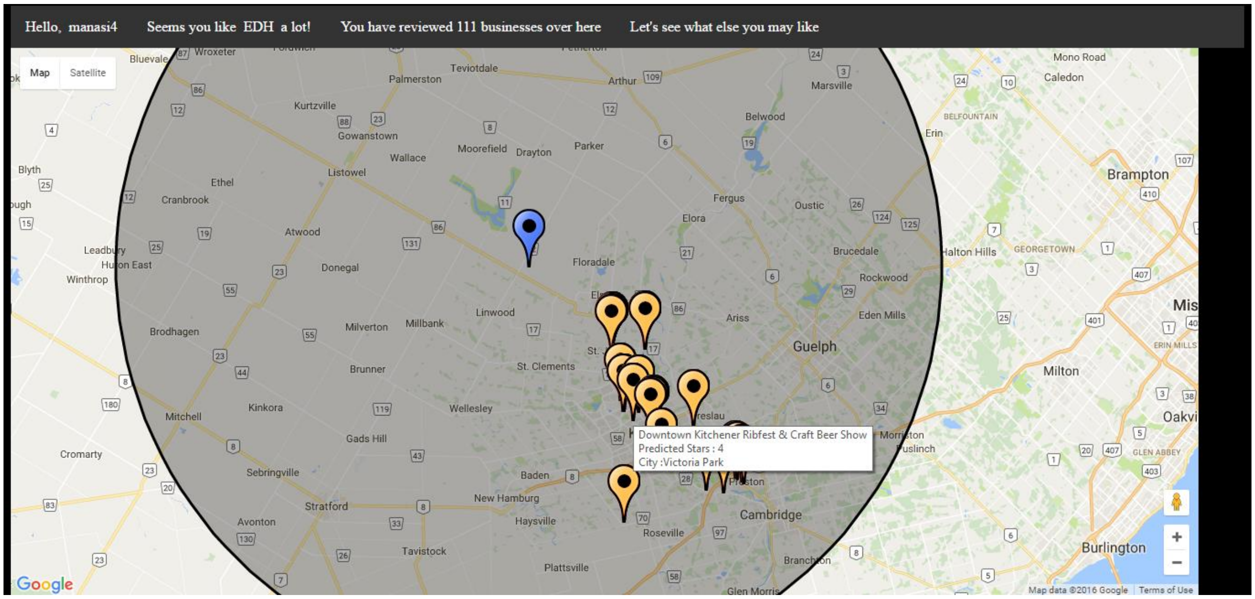
Chapter 5

5. Application

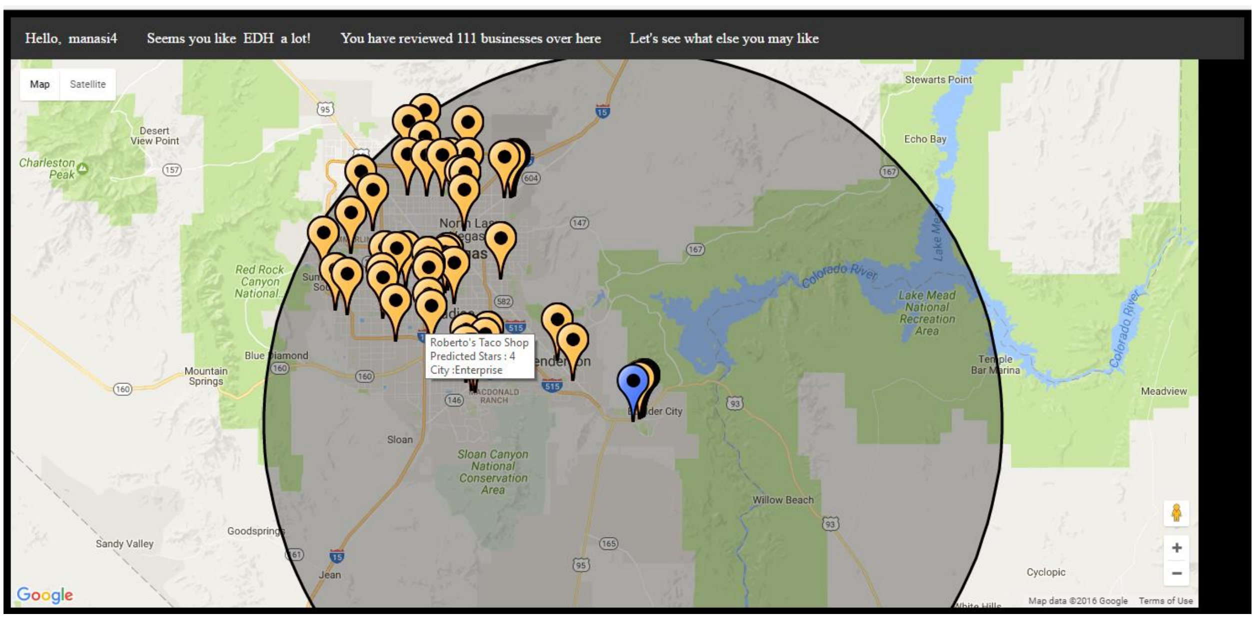
- Consider a user who had reviewed the food outlets in his region in Edinburgh, UK
- When the user logs in, he sees his rated businesses along with the sentiment score of his reviews and business average ratings
- This region is termed as hot spot of the user
- Google Maps API and Geometry Library were used



- When the user moves to different location, say Cambridge in US, he receives the predictions for the food businesses he has not reviewed yet
- These predictions will be based on businesses that he has reviewed in Edinburgh, UK



- The yellow markers show the new businesses
- Similarly, when he moves his location to Las Vegas, he sees a different prediction model



Chapter 6

6. Challenges faced

- **Selecting features:**
  Since there were 95 features for the businesses, it was a challenge to select the relevant features to predict with high accuracy. To overcome this problem, we analyzed each attribute and eliminated the features that had a lot of Null values and which were not relevant for Food related businesses.

- **Loading the data**
  We first faced this issue while loading the reviews data feed into R for data analysis since it was around 2gb. We tried increasing the heap memory of R but still it was not able to process. To overcome this problem we used the *fread* function of *data.table* library which is meant to import big data from regular delimited files directly into R, without any detours.

- **Processing the review data for sentiment analysis**
  To perform the sentiment analysis on each and every word of all the reviews, we developed a R code which was not able to process because of memory limits. Therefore, we implemented the sentiment analysis algorithm on Hadoop MapReduce on AWS EC2 instance.

- **Data standardization**
  Since the algorithm that we implemented assumes that all features are centered around zero and have variance in the same order, we were getting lower accuracy. To resolve this issue we used the *preprocessing* module of *scikit-learn* library for mean removal and variance scaling.

Chapter 7

7. Tools & Technologies Used

Technology	**IDE/ Framework**
Python	Spyder
R	R Studio
MapReduce	Eclipse/ Hadoop
Spring MVC	STS
Hadoop cluster	AWS EC2 instance
MongoDB	Mongo 3T Chef

Chapter 8

8. Conclusion

We were able to build a prediction model which is able to help a yelper by predicting the ratings for the new businesses at a new location based on his/her historical data. This way we were able to mirror his/her personal likings on a new geographical location and were able to map his model.

Chapter 9

9. Future scope

- All Business Categories
  We can use this model to predict the user rating for all businesses which are a part of the data set provided by Yelp
- Social Graph Mining
  We can analyze and understand the yelping pattern that can calculate the degree of similarity between the user and his friends. By this we can figure out which user is a trend setter for a particular business.
- #YELFIE
  According to the current '#Yelfie' trend we can analyze the sentiment of a user by image recognition which can be included as a feature in our current data set.

Chapter 11

10. References

1. Data Analytics using Yelp Data [Nevil Patel, Suraj Ponugoti, Doan H Nguyen]
2. LIBSVM: A Library for Support Vector Machines [Chih-Chung Chang and Chih-Jen Lin, Department of Computer Science, National Taiwan University, Taipei, Taiwan]
3. Applications of Machine Learning to Predict Yelp [Kyle Carbon, Kacyn Fujii, Prasanth Veerina]