A New Semantic Approach on Yelp Review-star Rating Classification

Shuang Wu
Department of Computer Science
University of Wisconsin-Madison
swu56@wisc.edu

Xiaodong Wang
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
xwang322@wisc.edu

Bozhao Qi
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
bqi2@wisc.edu

Abstract

This paper introduces a new semantic approach for yelp review star rating prediction. Our approach extracts feature vectors from user reviews to develop star prediction models. User review text contains detailed information about reviewers' experience, and directly reflects reviewer's satisfaction level. Our approach can extract sentimental words from review text, and convert these information into different feature vectors. Reviewer's personal preference may be extremely skewed from each other, to eliminate these effects, we use belief propagation methods to calculate review star probability distributions for different types of reviewers. Our machine learning algorithm predicts review star based on reviewers' preference and voting habit. We extract different feature vectors and apply them to several machine learning algorithms. To evaluate all the 2.2 million user reviews, we build spark system on three laptops. To achieve a better prediction accuracy, we perform sentiment analysis of reviews in terms of the number of positive, negative, negation words, and apply belief propagation methods to get rid of personal preference effects. Our system can evaluate 2.2 million data entries in less than two minutes and achieve an accuracy of 55%.

1 Introduction

With the spread of internet, crowd-sourced user review and social networking platform become more and more popular. For example, Yelp is a website allow users to rate local restaurants they have visited and TripAdvisor is place where you can find reviews for hotels, interesting places, etc. On these kinds of platforms, user can submit star ratings as well as review texts. Usually review texts provide much more detailed information than the quantitative metric of star ratings. Both of the star ratings and text ratings have their own pros and cons. The rating star provides a quick insight of the business but it is usually subjective. In contrast, the text review contains more detailed information about user experiences. In our project, we are going to explore how to translate the the highest granularity of details of text reviews into a metrics of a numerical rating.

Different users have different rating preferences, an easy going user would like to give 4 and 5 stars for most cases, while a picky user will give a 3 star even though he thinks this is a good restaurant. One extreme example is like this, for a 1-star rating, the review would be, "My pork chop with pomegranate sauce was actually quite tasty, so why a 1 star review? Rude, poor service. Our server was impatient, combative, and argumentative. Twice she came to the table and we were talking and didn't snap to immediately, she left and once gave us the 'Wrap it up' signal with her finger." The above rating star is subjective and biased to other consumers to view. Hence, we need to consider this when evaluating the review text.

In each review text, sentimental words are the keys to reflect reviewer's attitude towards their experiences. According to [3], sentimental analysis and opinion mining are studied, with a list of positive and negative words being provided. Hence, by counting the appearance of the words in different categories, we can have a sense of reviewer's satisfactory level. Then, combining with reviewer's personal preference information, our approach can provide an adjusted rating star.

2 Preliminary

2.1 Dataset

Our original datasets are obtained from Yelp data challenge. This datasets contain Tips, Users, Reviews, Businesses and Check-in tables. In short, 2.2M reviews and 591K tips are given by 552K users for 77K businesses. 566K business attributes refer to multiple attributes such as hours, parking availability, ambiance. etc. Social network of 552K users for a total of 3.5M social edges are presented.

2.2 Data cleaning and processing

We only use 4 out of 5 datasets from Yelp which includes Reviews, Tips, Users, and Businesses. This experiment is tested on the Ipython platform, and includes packages and tools imported from Pandas DataFrame, Numpy, Matplotlib, R wordcloud, etc. First of all, we rename the duplicate attribute names appears in each table for better view:

```
business_DF.rename(columns={'name': 'business_name', 'stars' :
'business_average_stars','review_count': 'business_review_count'},
inplace=True)
```

Secondly, we change data type from Object to int or float in each attribute columns for the convenience of later computation. e.g:

```
user_DF['user_average_stars'] =
user_DF['user_average_stars'].astype(float)
```

Thirdly, after scrutinizing all the tables in MySQL, we find that many instances show that some Yelp users wrote multiple tips and reviews for the same business entity on the same day. W.O.L.G, we decide to shrink the size of the whole

datasets by combining those tuples based on primary key of [business_id, user_id, date]:

review_DF1 = review_DF.groupby(['user_id','business_id', 'date'])['review_text'].apply(lambda x: ','.join(x)).reset_index()

review_DF2 = review_DF.groupby(['user_id','business_id', 'date'], as_index=False).mean()

review_DF3 = review_DF.groupby(['user_id','business_id', 'date'], as_index=False).mean()

Likewise, we do the same operations on Tips table and a left outer join on review_DF3 to maintain the size of the review_DF3 because the size of Reviews table is much bigger than size of Tips and Reviews table which contains the classification label [review_star]. After that, we create a new column by combing all the text documents from Reviews and Tips:

review_and_tip_DF = pd.merge(review_DF3, tip_DF1, on =['user_id', 'business_id', 'date'], how='left')

review_and_tip_DF["text"] = review_and_tip_DF["review_text"].map(str) + ', ' + review_and_tip_DF["tip_text"].map(str)

Fourthly, we generate our final table by inner joining Businesses, Users and aforementioned Reviews_Tips based on [user_id, business_id]. e.g.

business_user_review_and_tip_DF = pd.merge(user_review_and_tip_DF, business_DF, on =['business_id'], how='inner'

We realize this Yelp challenge datasets are only part of the real datasets so that we have to recalculate some of attributes in order to make our data consistent.

temp_DF = business_user_review_and_tip_DF.groupby(['user_id']).mean()['review_stars'].reset_index()

temp_DF.rename(columns={'review_stars': 'user_average_stars'}, inplace=True)

business_user_review_and_tip_DF = pd.merge (business_user_review_and_tip_DF, temp_DF, on =['user_id'], how='inner')

Finally, final table is cleaned by removing noise redundants:

business_user_review_and_tip_DF = business_user_review_and_tip_DF.drop (['attributes.Price Range','votes.cool', 'votes.funny', 'votes.useful'], 1)

This final table contains around 2.2 million tuples and each of them has 31 columns of attributes and 1 classification label. This table will be referred to later multiclass classification problem.

3 Exploratory Data Analysis

We set up a webpage for testing our new query functions, and two main tools are implemented during our experiment. Review star trends and Word cloud offer users a direct visualizations.

3.1 Review star trends

The final table is grouped by business_id and date, and we recalculate the average review star. Then Yelp users can clearly see a quality trend of this unique business entity by entering its bussines_id, Figure 1 shows different trends of three companies.

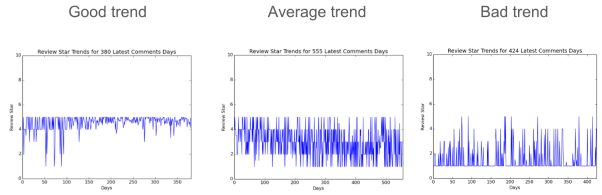

Figure 1. Business average review star trends.

A good trend for a business entity is that its average star is increasing from time to time.

3.2 Word cloud

We perform these experiments on Yelp real world datasets in which labels are on an integer scale from 1 to 5. We consider reviews with a score of 1 and 2 to be negative, scores of 4 and 5 to be positive, and score of 3 to be mediocre. After entering the business_id, three figures will be generated for Yelp users. This visualization gives users a direct judgment about some obvious tag words of this unique business entity.

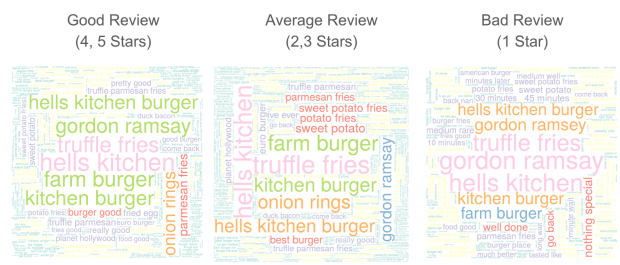

Figure 2. Word cloud for different user review texts.

From Figure 2, we could clearly see that the "onion rings" has a good rating and "sweet potato" is not such good. Yet, "hells Kitchen burger", "farm burger" and "gordon ramsay" are the most commonly words in all three pictures. Those words can be efficiently removed by tf-idf method in text processing.

4 Review Star Prediction

4.1 Machine learning algorithm

We split raw data into training and testing part by a 60:40 ratio. We choose multiple machine learning methods to train and test data. Methods are already defined in sklearn in Python library and we implement them in our iPython interface. We have tried multinomial Naive Bayes, Support Vector Machine, Random Forest, and Decision Tree. We use precision and recall as performances metrics of the different classification models, representing quality and quantity respectively of documents classified and ranging between 0 and 1 inclusive. They are defined as follows:

$$Precision = TPR = \frac{TP}{TP+FN} \tag{1}$$

$$Recall = PPV = \frac{TP}{TP+FP} \tag{2}$$

Where: Precision(c) = | {Yelp reviews with c star category} ∩ {Yelp reviews predicted as having c star category} | / {Yelp reviews predicted as having c star category}

Recall(c) =| {Yelp reviews with c star category} ∩ {Yelp reviews predicted as having c star category} | / {Yelp reviews with c star category}

4.2 Feature vector selection

We have two rounds of extracting feature vectors from reviews. As reviews are located in one column in the dataframe whose format is text, we have to preprocess them first. We have done some work such as removing numbers and punctuations, converting all letters to lowercase, filtering out those

stopwords and tokening eventually. By then we have a relatively clean list containing all words which are ready for selecting features.

The first round is simply tokening one word, two words and three words in the list. This is usually a general way to find some inner relationship inside the list. We will show the results in the evaluation part, here we briefly introduce what we have done to generate feature vectors.

The second round is introducing positive and negative word lists showing the sentimental aspect of reviews. The collections of positive and negative words is called sentimental words. We are interested in how many positive/negative words in one text. Based on the data we have collected, we created a few feature vectors such as positive word count, negative word count, sentiment word count, positive word percentage, negative word percentage and their ratio over each other. We hope them would be useful for our predication.

5 Scalability

In this project, we configure and set up a small cluster joined by three MacBook machines, each has 4 cores and 2 gigabytes to be dispatched. One of them serves as both NameNode and DataNode in order to take advantage of utilization rate and the small scale batch-processing. The other two will only serve as DataNode. Furthermore, the NameNode machine will also serve as Spark Master, and rest will to be the slaves in the whole cluster.

5.1 HDFS groundwork

We delve into the working mechanism of two closely related distribution file systems, the Google file system (GFS) and Hadoop distributed file system(HDFS) in order to tackle with the scalability issue. The HDFS is the open source version of the GFS. Both Systems are designed for batch-processing on a large data sets which could be up to several petabytes. In our case, though our data scale is not perfectly suited for the large distributed storage and computation framework, it could shed a new light on the the efficient parallel computation on our project[5].

The HDFS provides a file system at a user-level instead of a file system at VFS operating-system level. In addition to basic create, delete, open, close, read, and write operations, it also provides two special operations which are snapshot and append. Append operation resolves the issue of appending data concurrently to the same file with locking a file.

In HDFS, the file is splitted into multiple 64 MB blocks, and those blocks are stored and replicated on different DataNodes for the purpose of fault tolerance. In contrast to the 4 KB block size in traditional file system, the 64MB block size can significantly relieve TCP connection and storage pressure on NameNode's main memory[6].

The NameNode is responsible for FsImage, EditLog and In-memory FS metadata. The FsImage is an checkpoint image stored in B-tree to preserve the system state periodically and recreate metadata on cluster crash or startup without reading the entire EditLog[1]. The Editlog is in charge of recording all the changes to any HDFS metadata such as file creation, addition of new blocks to files, file deletion, changes in replication, etc. The metadata is only stored on the NameNode's main memory, and it includes the whole file system name space and name-to-block mappings.

The DataNode is responsible for simple process requests from client to create, delete, write, read blocks or replicate blocks. DataNodes also will send the block reports and beatheart messages back to NameNode for maintaining good functioning and connection[2]. Due to space limit, we provide details about HDFS setup in Appendix.

5.2 Spark groundwork

We choose Spark instead of Hadoop MapReduce in terms of three high-performance features. First of All, Spark offers us a more flexible computation framework while MR has to impose a rigid computation model which is confined to only two-stage process: Map & Reduce. In turn, application has to run MR multiple passes to get final results. More passes means more cost for disk input/output and replication. Secondly, Spark provides high-level interface to users and also supports SQL queries, graph processing, and machine learning which is adopted in our case. Thirdly, unlike MR, the Spark will cache the intermediate results in memory, and most machine learning algorithms are of complex and multi-iterative stages, so that this cache mechanism will significantly accelerate machine learning algorithm performance about 20 times faster than Hadoop[9]. In driver program which is linked with Spark API, the SparkContext will be created. The SparkContext is an connection to Spark cluster manager which will allocate resources to slave nodes and initiate executors in each of them. An executor is a process which runs an instance of JVM, and each executor on different slave node will store cache and run tasks which are application code of transformation and action[10].

Resilient Distributed DataSet(RDD) is the data form in the Spark. In our case, the input HDFS data is organized into RDD. RDD provides an interface to manipulate data by transformations and actions. Transformations include map, filter, groupByKey, and reduceByKey, and basically transform an input RDD to a new RDD. Actions include reduce, grab samples, and write to file, and basically output a value from RDD[8]. Due to space limit, we provide details about Spark setup in Appendix.

5.3 MLlib

MLlib is Apache Spark's scalable machine learning library. After comparing Naive Bayes, Decision Tree and Random Forests in our experiment, Decision Tree shows the best classification results among all the classifiers. The ML algorithms on Spark cluster are implemented differently from the algorithms on single machine. In this section, we will dig out more optimization details about our best classifier, Decision Tree. We use Gini (Equation 3) as our impurity measure for classification.

$$Gini\ impurity : \sum_{i=1}^{C} f_i(1-f_i) \tag{3}$$

where f_i is the frequency of $label_i$ at a node and C is the number of unique labels. Dataset D is partitioned into D_{left} and D_{right} by splitting s, the basic info gain function is as

follow:

$$IG(D,s) = Impurity(D) - (\frac{N_{left}}{N} Impurity(D_{left}) + \frac{N_{right}}{N} Impurity(D_{right})) \quad (4)$$

There are three optimizations for splitting candidates on Spark[4].

1. **Sampling:** When the node is an continuous feature, sorting this feature value is extremely expensive for enormous datasets. Thus this implementation only computes an approximate set of split candidates by performing a quantile calculation over a sampled fraction of the data.
2. **Binning:** When the node is an categorical feature and if it has M possible disordered categories, then it has $2^{(M-1)}$ - 1 possible ways to split in multiclass classification whenever possible. If this feature is ordered, then it only has M -1 ways to split.
3. **Level-wise training:** The tree building on single machine is sort of depth-first approach, Yet this approach need to gather data from each subnodes, and it is impossible to be implemented owing to our oversized data. Thus we consider width-first approach on the Spark cluster and we only need to calculate the statistic parameters whenever we iterate the datasets. The number of iterations will equal to tree depth. Then it only requires L passes instead of 2L - 1 for full tree. For instance, 4 passes instead of 15 for Depth 4, and 10 passes instead of 1023 for Depth 10.

Due to space limit, we provide python code snippet of our Decision Tree in Appendix.

6 Belief Propagation

One phenomena we have observed from machine learning classification analysis is that there are always some users who prefer to give some comments with mostly nice/tough words but the ratings are fairly low/high. This is probably because the users have a personal habit of writing reviews in skewed proportion of sentimental words while his rating is prone to be opposite. There would be another case for example one user is experiencing a hard time when he has his dinner/lunch at the restaurant, but believes it is a nice restaurant so he still gives a good rating. Whichever reason, it decreases the precision and recall of matching review words with rating star. So this would be our breakpoint of improving accuracy of overall predication.

Given by suggestion from the course instructor, we study the literature about belief propagation (BP) method and hope this method could help in solving this problem[7]. Belief propagation method is developed for exploiting the way in which the global function factors into a product of simpler "local" functions, each of which depends on a subset of the variables. Such a factorization can be visualized using a factor graph, a bipartite graph that expresses which variables are arguments of which local functions(Figure 3).

To have an easy and general understanding about BP method, we generate a graph explaining how theorem is applied (Figure 4). Introducing such as $m_{ij}(x_j)$, which can be intuitively be understood as a "message" from a node i to a node j about what state j should be in. The message $m_{ij}(x_j)$ is a vector of the same dimensional as x_j, with each component being proportional to how likely node i thinks it is that node j will be in the corresponding state.

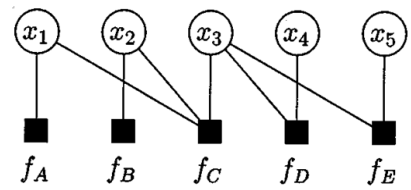

Figure 3. Bipartite graph of factorization.

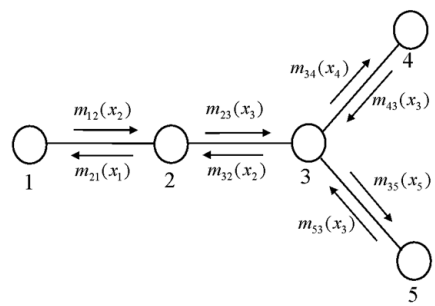

Figure 4. BP message propagation illustration.

In our case, we have nodes "user" and "restaurant" between which linked with reviews and stars as "messages"[11]. Right now we focus on star ratings. The message exists when some users have visited some restaurants and left reviews/stars and therefore two node linked with an edge. Every user has an individual rating distribution for every restaurant he went. He/she can give different distribution for restaurants he went. We could interpret the data we have in another way, for example, user A gives restaurant B a five-star rating. It looks good, which helps us believe more in that B is a "good" restaurant or at least an above-level restaurant. However, it really depends on what kind of user A is. Giving an extreme case, if A gives all restaurants he/she has visited all five stars, we would only say A is a "too easy-going" user which is really hard to tell the differences within all the restaurants he/she has visited. So in this case, this five-star message between A and B is not that trustworthy and it will be not as worth as a general user whose average user star is around three giving a five star rating to a restaurant. So we say, in BP algorithm, the belief at a node i is proportional to the product of the local evidence at that node ($\varphi_i(x_i)$) and all the messages coming into node i. We use b as the total belief in that node.

$$b_i(x_i) = k\phi_i(x_i) \sum_{j \in N(i)} m_{ji}(x_i) \quad (5)$$

Where k is a normalization constant (the beliefs must sum to 1) and N(i) denotes the nodes neighboring i. The messages are determined self-consistently by the message update rules:

$$m_{ji}(x_i) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{i,j}(x_i,x_j) \sum_{k \in N(i)\setminus j} m_{ki}(x_i) \qquad (6)$$

Note that on the right-hand-side, we take the product over all messages going into node i except for the one coming from node j. So taking our data for example, to fairly evaluate a restaurant real level, we can take a look at every user who has visited it and the corresponding star this user has given. But it is not a simple aggregate operation like mean or average, rather taking considerations of others ratings those users have given. For example, to tell "Taco Bell" is possibly above average level or not, we need to see every user who has given a rating to "Taco Bell" and dig into what stars those users have given to other restaurants.

It is easy to convince oneself, and to prove, that BP in fact gives the exact marginal probabilities for all the nodes in any singly-connected graph. In a practical computation, one starts with the nodes at the edge of the graphs and only computes a message when one has all the necessary messages. In general, it is easy to see that each message needs only to be computed once for singly connected graphs. That means that the whole computation takes a time proportional to the number of links in the graph, which is dramatically less than the exponentially large time that would be required to compute marginal probabilities naively. As mentioned before, the point of view illustrated by this example suggests that belief propagation is a way of organizing the "global" computation of marginal beliefs in terms of smaller local computations.

What we are interested is that BP will give a good marginal function or distribution for all nodes in the graph. We can separate review and star into two different graphs, we will just simply discuss star graph. This is a bipartite graph whose nodes on one represents one user and nodes on the other side each represents one restaurant. If there exists a star left by one user to one restaurant, then they are connected in this graph. We will not consider the case that one user visits one restaurant multiple times. That would be a little complicated to deal with which we will talk about in discussion section. For now, we assume one user at most visits one restaurant at a time.

We write the codes in Python to implement BP algorithm. To set up a test case, we constructed a simple graph which has three "user" nodes and three "restaurant" nodes (Figure 5). We connect every user node with every restaurant node which implies that every user has visited every restaurant. We deliberately assign different distributions for each user. One of them gives relatively low score to all restaurants which we assume it is a "picky" or "rigid" user while we define another user as "average" and the last one as "easy-going" or "nice". As you can probably tell from the word, "easy-going" user gives relatively high score to restaurants and "average" is between "picky" and "easy-going". We have done similar things to "restaurant" side. We also have one "good service" or "high level", one "so-so service" and one "horrible service" or "low level" . Each user at each iteration sends three distributions to all three restaurants, each distribution is over 5 different values which represents how much likely this user would evaluate this restaurant. Each restaurant node would receive one distribution message from each user in iterations. The iteration would carry on as long as no user nodes receive new messages. Once the convergence condition is satisfied, the result of user node is stable and returns a distribution reflecting his/her true evaluation distribution.

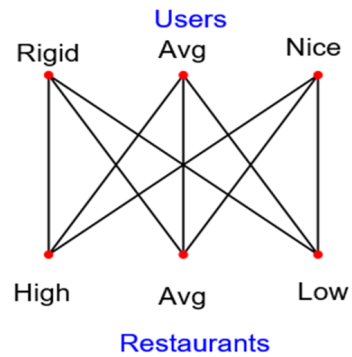

Figure 5. BP Test Case Illustration.

We calculate every user marginal distribution in the graph, and the following results are their distributions (Figure 6 and 7). It is interesting to see that if taking simple average score among each user has given, the distribution is a quasi-normal distribution. However, after BP calculation, the profile is prone to be an exponential curve which means extreme users like to give extreme scores more than what we have seen generally. This is interesting to watch as BP will give a fair evaluation by getting rid of influences from skewed data. We will discuss how it is implemented in our real data in later section.

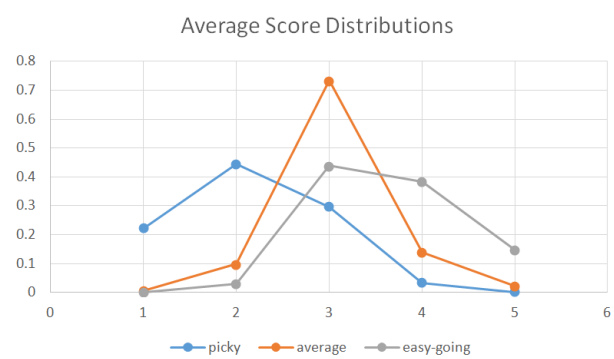

Figure 6. Average star value of each user.

7 Evaluation

7.1 Prediction Accuracy

As mentioned before, we use NLTK package to tokenize all the text documents, and remove stopping words, punctuation and do lowercase transformation in the final table. After that we calculate the positive and negative word frequencies. We vectorize our final table and apply three major classifiers onto the subset of the final table. This subset only randomly samples 20K tuples out of 2200K tuples.

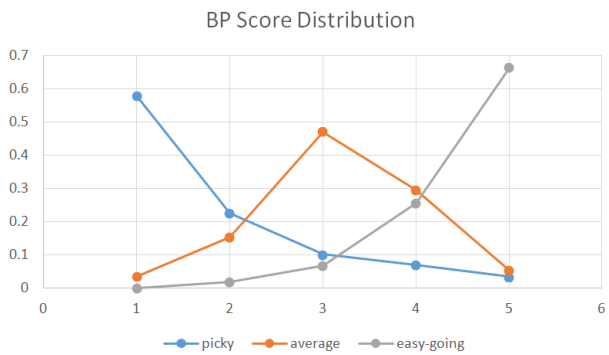

Figure 7. BP marginal distribution of each user rating star.

All in all, Decision Tree and Random Forest show us the best accuracy on the subset so far, which is up to 52% (as shown in Figure 8):

```
MODEL: Random Forest (100 Learners)

Precision: 0.524518112825
Recall: 0.522125
F1: 0.51985591112
Accuracy: 0.522125

Classification Report:
             precision    recall  f1-score   support

     1 star       0.74      0.68      0.71       829
     2 star       0.45      0.36      0.40       866
     3 star       0.36      0.31      0.33      1426
     4 star       0.48      0.60      0.54      2597
     5 star       0.63      0.57      0.60      2282

avg / total       0.52      0.52      0.52      8000


Precision variance: 0.091085

Recall variance: 0.101217
```

Figure 8. Random Forest prediction result.

The runner-up is the Support Vector Classification on the subset. It gives us accuracy of 42% (as shown in Figure 9).

The last one is the Multinomial Naive Bayes, which only presents us accuracy of 33% on the subset (as shown in Figure 10).

```
MODEL: SVC

Precision: 0.442193249157
Recall: 0.429375
F1: 0.401477354871
Accuracy: 0.429375

Classification Report:
             precision    recall  f1-score   support

     1 star       0.68      0.40      0.50       829
     2 star       0.35      0.12      0.18       866
     3 star       0.31      0.13      0.19      1426
     4 star       0.38      0.71      0.50      2597
     5 star       0.54      0.42      0.47      2282

avg / total       0.44      0.43      0.40      8000


Precision variance: 0.095435

Recall variance: 0.236549
```

Figure 9. SVC prediction result.

Although none of them above exceeds accuracy of 60%, the classification result is still awesome considering that this is five-classes classification problem. Thus, the Decision Tree & Random Forest will be adopted on our further 2.2 millions dataset. The Decision Tree on Spark gives us best result up to accuracy of 55%. This classification performance is boosted slightly in terms of larger training dataset. The final result is shown in Figure 11.

```
MODEL: Multinomial Naive Bayes

Precision: 0.326300947695
Recall: 0.242
F1: 0.234186014209
Accuracy: 0.242

Classification Report:
             precision    recall  f1-score   support

     1 star       0.15      0.75      0.25       829
     2 star       0.16      0.09      0.11       866
     3 star       0.24      0.20      0.22      1426
     4 star       0.38      0.12      0.18      2597
     5 star       0.44      0.29      0.35      2282

avg / total       0.33      0.24      0.23      8000


Precision variance: 0.069120

Recall variance: 0.293621
```

Figure 10. Multinomial Naive Bayes prediction result.

7.2 BP improvement

We have already implemented the test case, which makes us feel the real BP distribution is different from simply taking average of the user rating statistics. Here we try to work on setting up a BP graph using real value in database. One thing we realize when we try to set up a realistic model is that there is no way to completely simulate a real bipartite graph involving all users and restaurants. The reason is simple, because each user will send a message containing a 5 by 1 list to every restaurant he/she visited. This is a fairly small amount of information, but for some users in the database, they have visited almost 1000 restaurants. Thus the matrix grows to 5^{1000} which is almost more than the largest computer can handle on the planet. So there is no way to do this at best scenario. Instead we simply reduce our users into particular categories because this implementation is trying to differentiate different users having different rating habit. What we are doing here is testing whether this method will improve accuracy of predicting some extreme cases and we will propose some more accurate models in our discussion part.

```
16/05/03 13:11:55 INFO DAGScheduler: ResultStage 16 (count at
/Users/patron/Downloads/spark-1.6.1-bin-hadoop2.6/./cs838/decision_tree.py:29) finished in 28.744 s
16/05/03 13:11:55 INFO DAGScheduler: Job 10 finished: count at
/Users/patron/Downloads/spark-1.6.1-bin-hadoop2.6/./cs838/decision_tree.py:29, took 28.754599 s
Accuracy = 0.556122089442
```

Figure 11. Decision Tree prediction result on large dataset.

As to verify whether the BP method is useful, we make a simple case similar to test case. That will only working on those extreme users, meaning that their rating average is away from most users. We want to focus on users whose review counts is more than a minimum statistically stable number. That means this user has given more than a chosen number N. Same to restaurant, we want restaurant is at least visited by a fair amount of times, not judged by very few users. So taking consideration of everything, we decide to take users who have given more than 30 reviews and restaurants which have been given reviews for more than 30 times. This reduces our data by a fairly big percentage. As we only calculate a 3 by 3 model, we need to reduce our data to those extreme users. We define as follows: ”Nice” users whose

average score is more than 4.0, ”average” users whose average score is between 3.1 and 3.5 and ”tough” users whose score is below 2.5. Similar things is applied to ”restaurant” as well: ”good” restaurants are those average score above 4.1 and ”average” restaurants are those between 3.2 and 3.5, while ”bad” restaurants are below 2.5.

After classifying data into 3 groups, we join them by inner join. Then we calculate the percentage of each star given by each user and use that as probability table. After several rounds of iteration, we get the BP distribution of each user and we calculate expected value of each kind of user and name this value as ”BP average stars”.

We add one more column in dataframe named ”BP average stars” with the values mentioned above. We hope this value would help in justifying closer rating expectation with review text. We use Random Forest, Naive Bayes and Support Vector Machine as methods for analysis. As every test we have done, Random Forest gives the best accuracy, and we will discuss the results from Random Forest.

We create multiple sets of feature vectors as we are not sure which set of them gives us the best results. First, we include the number of positive words, the number of negative words, and the number of all sentimental words, positive word proportion, negative words proportion and their ratio. This gives a precision 36% and recall 38.78% which is not very good, but as all the attributes are in original database which means we have not used new BP average stars, this is regarded as a control experiment. We plot the confusion matrix for a better illustration about how actual label and predicted label are distributed (Figure 12).

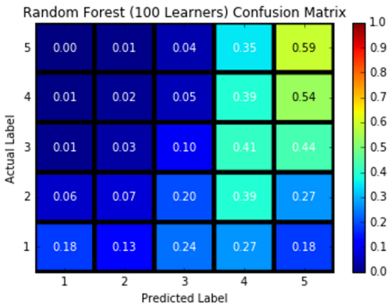

Figure 12. Confusion matrix for first round prediction.

We find that by reducing redundant attributes such as positive words count and negative words count and number of all sentimental words will help to increase precision to 37.32% and recall to around 40%, though not much improvement (Figure 13).

We decide to use user average stars listed in database as a trial. This shows a little improvement in precision to 39.532% and recall to 40.12%, still not outstanding improvement, which leads to last round where we add BP average stars (Figure 14).

We decide to replace the user average stars with BP average stars which might show some benefits in predicting user habit, and it does show somewhat improvement on the precision and recall. The precision goes up to 42.71% and recall now is 44.95%, both of which improve a little. This gains

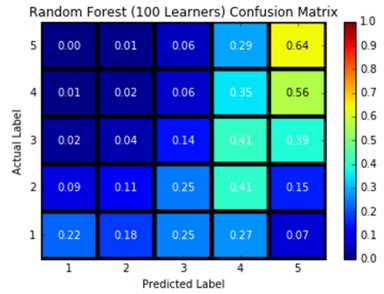

Figure 13. Confusion matrix for second round prediction.

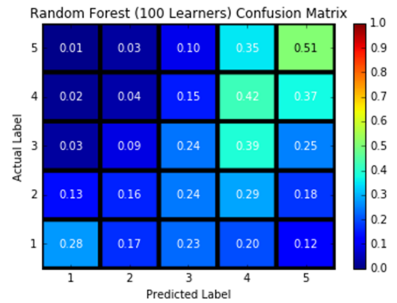

Figure 14. Confusion matrix for third round prediction.

our confidence that BP is a possible solution, however, we are looking for more improvement (Figure 15).

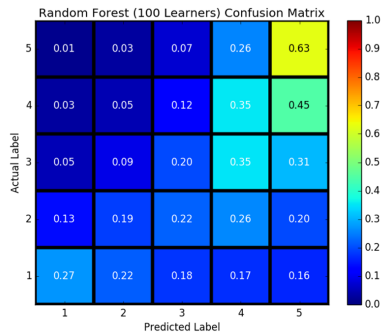

Figure 15. Confusion matrix for forth round prediction.

Finally, the attribute ”user average stars” gives around 3% improvement in precision and BP increases both precision and recall in fourth round, we decide to include both of them in last trial and it does show a good trend. This time the precision is 48.91% and recall is 51.45%, and both of which are about 10% higher than before using BP method. This gives us confidence that BP will help predicting skewed users taking consideration that our model has simplified a lots of factors compared with real data (Figure 16).

8 Discussion

8.1 Limitation

8.1.1 About BP

As mentioned before, our computation resources are limited. So implementing a complete calculation for each user in the database is not possible for this course project. Actually it is even impossible for anywhere in real world. However, an alternative solution is using fine mesh in the rating

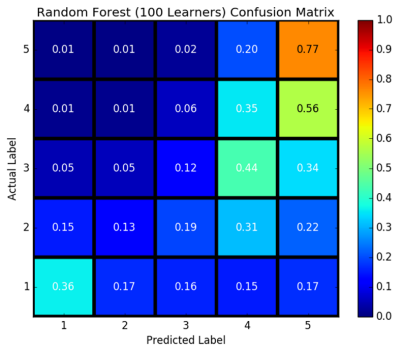

Figure 16. Confusion matrix for fifth round prediction.

for categorization. For example, we separate our user average star and business average star as 0.2 step as one sub category, and we will have 21 different levels for both users and restaurants. Then we take frequent users and frequent restaurants into consideration, and take average score for each user node by its connected restaurant node. We would have at most 5^{21} elements in our matrix which is possible to calculate for company owning super large computation resources ($4 * 10^5$ GB RAM). If this is too difficult, we can also have 0.25 step as a coarse alternative approach which asks for 640 GB memory requirement which is easy to access for industrial company. In any case, resulting BP average score would be useful for predicting extreme user rating habit.

8.1.2 About feature vector

We have a strong feeling that our feature vectors have not fully reflected every aspect of what users want to express in their review texts. Our feature vectors are based on extracting the sentimental words, and calculating related parameters. However, this still gives us not a quite excellent satisfactory result which means that something deep needs to be dig out, this gives us a heading direction for future work.

8.2 Future Work

We feel like there is a lot of work on extracting useful words from user reviews. Not only taking out positive/negative words, but also digging more into their relationships. This requires a lot of time for studying field of natural language processing. Some other possible work would locate in connecting other columns in database for this whole analysis. Right now we only focus on relationship between user review and user rating. But there would be many other factors influencing the final rating, for example, time and location. We have no time digging into those hidden information but it is fun to run some simple queries first to take a quick look at overall relationships between the location of restaurants and their overall rating levels. This would involve a much more analysis about the overall database.

9 Conclusions

In this project, we have performed different operations to clean the raw data using MySQL and IPython. We also present two visualized tools for Yelp users and after that we extract multiple feature vectors from clean data and explore different machine learning algorithms to find the best prediction model and finally we set up a computing platform with a large computing power using spark to evaluate more than 2 million tuples in less than 2 minutes. The results show the effectiveness and reliability of our approach. To deal with difficulties in classification for extreme users, we import BP method and test on simple case for verification. This gives us promise that it would be a better way for accuracy improvement. Then we apply it on our data and it shows around 10% improvement on classification accuracy after applying our simpler model, it still improves prediction accuracy around 10% which is promising. Other than academic achievements, our problem solving and team working skills also improved.

10 References

[1] Cluster mode overview. *http://spark.apache.org/docs/latest/cluster-overview.html*.

[2] Job scheduling, spark 1.6.1 documentation. *http://spark.apache.org/docs/latest/job-scheduling.html*.

[3] Opinion mining, sentiment analysis, and opinion spam detection. *https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html*.

[4] Scalable distributed decision trees in spark mllib. *https://spark-summit.org/2014/talk/scalable-distributed-decision-trees-in-spark-mllib*.

[5] D. Borthakur. Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[7] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[8] P. Nathan. Intro to apache spark. *http://stanford.edu/ rezab/sparkclass/slides/itas-workshop.pdf*.

[9] S. Penchikala. Big data processing with apache spark. *http://www.infoq.com/articles/apache-spark-introduction*, 2015.

[10] S. Ryza. Apache spark resource management and yarn app models. *https://blog.cloudera.com/blog/2014/05/apache-spark-resource-management-and-yarn-app-models/*, 2014.

[11] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

A Environment Setup and Code Snippet

A.1 HDFS Setup

Key steps for our HDFS Setup:

1. /.ssh/ ssh-keygen *t* rsa & disseminating authorized_key
2. /etc/hosts/configure master&slaves IP:
   xxx.xxx.xxx.xxx master xxx.xxx.xxx.xxx slave
3. download java runtime enviroment
4. Configure Home Path in /.bash_profile and /.bashrc for each node:
5. ./bin/hadoop NameNode −format & run ./sbin/startdfs.sh
6. ./bin/hdfs dfs mkdir /CS838_Spark
7. ./bin/hadoop fs −put Users/patron/Desktop/final.txt
8. ./bin/hadoop fs −get hdfs: //master:54310//spark
   /output.txt/Users/patron/Desktop

A.2 Spark Setup

Key steps for our Spark Setup:

1. ./sbin/startmaster.sh
2. ./sbin/start-slave.sh -m 2G -c 4 -h spark: //
   10.141.167.255:7077
3. ./bin/spark-submit –master
   spark://10.141.167.255:7077
   –total-executor-cores 4 –executor-memory 2G

A.3 Decision Tree

The main step in our Decision Tree python code:

```
model = DecisionTree.trainClassifier(trainingData, numClasses=7,
categoricalFeaturesInfo={},impurity='gini', maxDepth=5, maxBins=32)
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
accuracy = labelsAndPredictions.filter(lambda (v, p): v == p).count() /
float(testData.count())
```