

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 **extract** feature vectors from user reviews to develop **user-review** star prediction models. User review text contains detailed information about **reviewers** experiences, and directly reflects reviewers satisfaction level. Our approach can extract sentimental words from review text, and convert these information into different feature vector. Reviewers 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 **will** predict review star based on **reviewers** preference. 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 **website** allow users to rate local restaurants they **visited** and TripAdvisor is place where you can find reviews for hotels, interesting places, etc. **In** these **kind** of platforms, user can submit star ratings as well as review texts. **And** **review** texts for provides 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** 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 **quick usefulness 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 **following**, "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 didnt snap to immediately, she left and once gave us the 'Wrap it up' signal with her finger." The above rating star is extremely subjective and biased to other **consumer** to view. Hence, we need to consider this when evaluating the review text.

In each review text, sentimental words are the keys to reflect reviewers attitude towards their experiences. According to [3], sentimental analysis and opinion mining are studied, **and** a list of positive and negative words **have been** provided. Hence, by counting the appearance of the words in different categories, we can have a sense of reviewers satisfactory level. Then, combining with reviewers 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 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 shows that some Yelp users wrote multiple tips and reviews for the same business entity on the same day. **Thus** 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 do a left outer join on review_DF3 and maintain the size of the review_DF3 because the size of Reviews table is much bigger than size of Tips and Reviews table contains the classification label [review_star]. After that, we create a new column by union 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 redundant: 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 the average review star will be calculated. Then Yelp users can clearly see a quality trend of this unique business entity by entering its business_id, Figure 1 shows different trends of three companies.

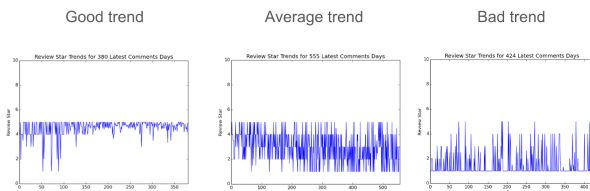


Figure 1. Business average review star trends.

The Good trend might show that this business entity is getting better and better even though its average review star is not such high.

3.2 Word cloud

We performed this experiments on Yelp real world datasets in which labels are on an integer scale review_stars from 1 to 5. We considered 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 give user a direct judgment about downside and upside of this unique business entity.

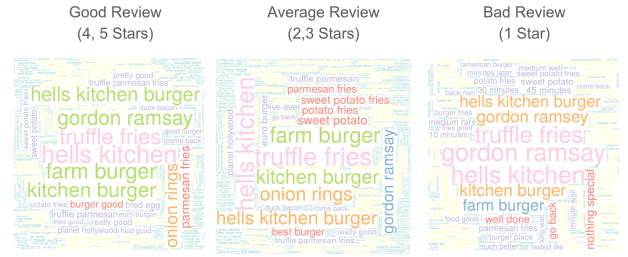


Figure 2. Word cloud for different user review texts.

From Figure 2, I 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.

4 Review Star Prediction

4.1 Machine learning algorithm

We split raw data into training and test data by a 60:40 ratio. We choose multiple machine learning methods to train and test our data. Methods are already defined in sklearn in Python library and we implement them in our iPython interface. We tried multinomial naive bayes, support vector machine, random forest, decision tree and logistic regression. Then 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:

$$Precision = TPR = \frac{TP}{TP + FN} \quad (1)$$

$$Recall = PPV = \frac{TP}{TP + FP} \quad (2)$$

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 review. As reviews are located in one column in the dataframe whose format is text, we have to preprocessing them first. We have done some work such as removing the numbers and

punctuations, converting all letters to lowercase, filter out those stopwords and token eventually. Right now 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 list which have been introduced before. The collections of positive and negative words is called sentimental words. We are interested in how many positive/negative words in one text and their ratio over each other. 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 predicate.

5 Scalability

In this project, we configure and set up a small cluster joined by three MacBook machines, in which each of machine has 4 cores and 2 gigabytes to dispatch. One of them serve as both functionalities of 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, one of the other two DataNodes will serve as a Spark Master, and rest will 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. However, it could shed a new light on the efficient parallel computation on our project[5].

The HDFS provides file system interface at a user-level instead of file system calls under 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 resolve the issue of concurrent append data 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 NameNodes 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 NameNodes 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. And DataNodes also will send the block report and heartbeat message 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 are choosing 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 input/output disk and replication. Secondly, Spark provide high-level interface to users and also supports SQL queries, graph processing, and machine learning which is adapted 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[8]. 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 is just a application code of transformation and action[9].

Resilient Distributed DataSet(RDD) is the data form in the Spark. In our case, the input data HDFS 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[7]. 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 trees and Random Forests in our experiment, decision trees 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 in our tree.

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

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:

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

There are three optimization 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 method, Yet this method need to gather data from each subnodes, and it is impossible to be implemented owing to our over sized data. Thus we consider width-first method on the Spark cluster and we only need to calculate the statistic parameters whenever we iterate the datasets. The number of iterations will equals 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 predicate analysis is that there are always some users who prefer to give some comments with most nice/tough words but the rating is fairly low/high. This is probably because the users have a personal habit of giving reviews in skewed proportion of sentimental words while his rating is prone to be opposite. There would be other reason such as the user is experiencing a hard time when he had his dinner/lunch at the restaurant, but he stills think 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 predicate.

Given by suggestion from the course instructor, we search the literature about belief propagation (BP) method and hope this method could help in solving this problem. 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).

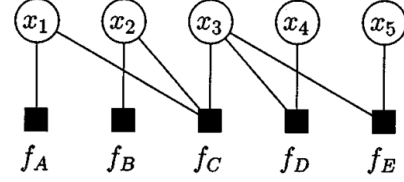


Figure 3. Bipartite graph of factorization.

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 $m_{ij}(x_j)$ will be 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.

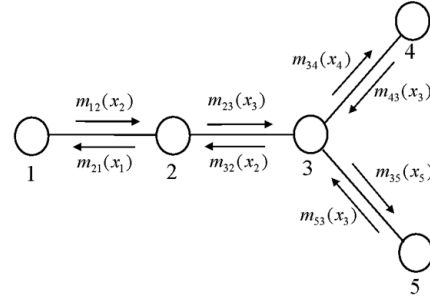


Figure 4. BP message propagation illustration.

In our case, we have nodes "user" and "restaurant" between which linked with reviews and stars as "messages". Right now we focus on star ratings. The message exists when some user have visited some restaurant and left a review/star there and therefore two node linked with an edge. Every user has an individual rating distribution to every restaurant he went. He/she can give different distribution form to 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 should sound good, which helps us believe more in 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, then we would only say A is a "too easy-going" user which is really hard to tell the differences within the restaurants he/she has visited. So in this case, this five star between A and B is not that trustworthy and it will not be as worth as a general user whose average star is around three gives 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 ($\phi_i(x_i)$) and all the messages coming into node i , b_i is total belief in that node.

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

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:

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

Note that on the right-hand-side, we take the product over all messages going into a 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 **user** has given. But it is not simple aggregate operation like mean or average, rather taking considerations of others ratings those users have given. For example, to tell "Taco Bell" is **mostly** satisfactory 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 ~~rating to "Taco Bell" given to~~ others restaurants.

It is easy to convince oneself, and to prove, that BP is 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 available all the messages necessary. In general, it is easy to see that each message need only **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.

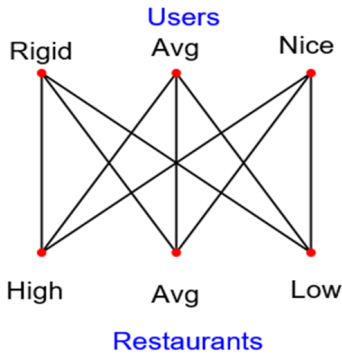


Figure 5. BP Test Case Illustration.

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 **which** nodes on one side each represents one user and nodes on the other side each represents one restaurant. If there exists a star left between one user and one restaurant, then they are connected in this graph. We will not consider the case that one user **visited** 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 **left** one **star** to one restaurant.

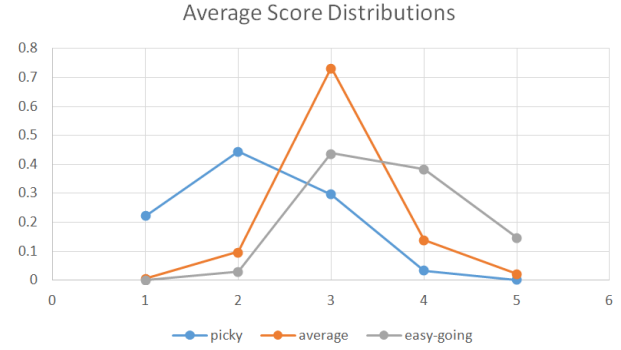


Figure 6. Average star value of each user.

We wrote 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 **defined** 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 as "horrible service" or "low level". Each user at each iteration sends three distributions to all three restaurants, each distribution is a distribution over 5 different values which represents how **much** this user would evaluate this restaurant. Each restaurant node would receive one **distribution** from each user in iterations. The iteration would carry on as long as each time what user node receives from all other nodes is not **the same**. Once the convergence condition is satisfied, the user node result is stable and returns a distribution reflecting his true evaluation distribution.

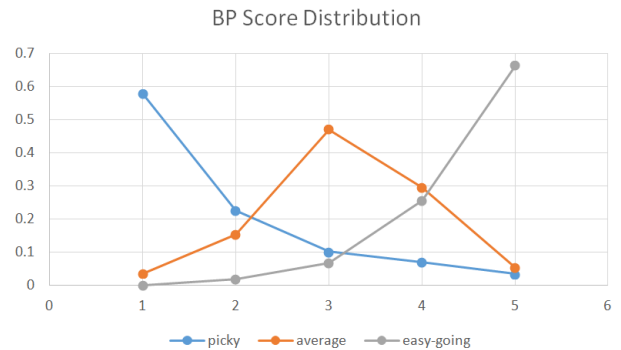



Figure 7. BP marginal distribution of each user rating star.

We calculate every user marginal distribution in the graph the following results are their distributions (Figure 6 and 7). It is interesting to see that if taking simple average **among each user**, the distribution is a quasi-normal distri-

bution. However, after BP calculation, the profile is prone to be an exponential curve which means **users** like to give extreme score **more chance**. This is interesting to watch as using BP will give a fair evaluation **getting** rid of influences from skewed **restaurants visited**. We will discuss how it is implemented in our real data in later section.

7 Evaluation

7.1 Prediction Accuracy

We use NLTK package to tokenize all the text documents in  final table, and remove stopping words, punctuation and do lowercase transformation on them. 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.

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 exploration. The Decision tree gives us best result up to accuracy of 55%. The 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 before, making us feeling the real BP distribution is **different** 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 **real value** is that there is no way to completely simulate a real bipartite graph involving all users and all restaurants. The reason is simple, each user will send a message **contains** a 5 by 1 **values** to every restaurant he/she visited. This is a fairly small amount of information, but for some users in the database, he/she has 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 user into particular categories because this implementation is trying to differentiate different users having different rating habit. What we are doing here is **test** whether this method will improve **that** and we will propose some more accurate model 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/jcs838/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/jcs838/decision_tree.py:29, took 28.754599 s
Accuracy = 0.556122089442
```

Figure 11. Decision Tree prediction result on large dataset.

As to verify the BP method is **working**, we make a simple case similar to test case, only working on those extreme users, **which mean their rating average is skewed**. We want to focus on users whose review counts **has been** more than **at least a certain number which is statistically stable**, **which means** this user **would have** given more than a chosen number N. Same to restaurant, we want restaurant we are studying is at least visited by a fair amount of times, not **simply** judged by very few users. So taking consideration of everything, we decide to take users who have given more than 30 reviews and for restaurants which have been **written a review** for more than 30 times. This reduces our data a fairly big percentage. As we only calculate a 3 by 3 model, we need to reduce our data **more on only** those extreme users. We define as follows: "Nice" users whose average score is more 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 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 **classify** 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 **function** 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 **regard** this value as "BP average stars".

We add one more column in dataframe named "BP average stars" with the values mentioned **before**. 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 **did**, random forest gives the best accuracy, we discuss the results from random forest.

We created multiple set of feature vectors as we are not sure which set of them gives us the best results. First, we include number of positive words, number of negative words, 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 distributed (Figure 12).

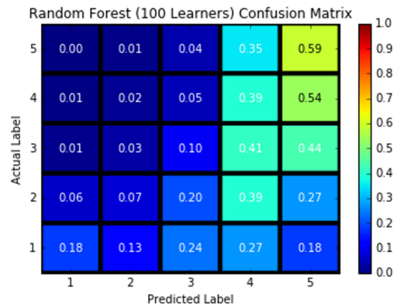


Figure 12. Confusion matrix for first round prediction.

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

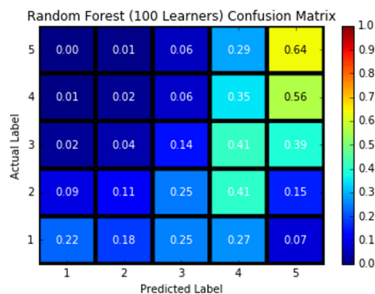


Figure 13. Confusion matrix for second round prediction.

We decided 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 **obvious** improvement, which leads to last round which we add BP average stars (Figure 14).

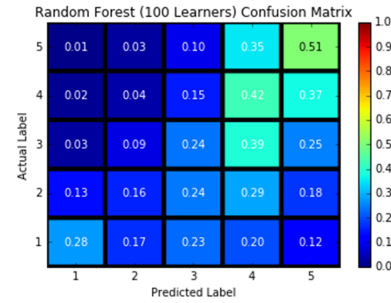


Figure 14. Confusion matrix for third round prediction.

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 on the precision and recall. The precision goes up to 42.71% and recall is now 44.95%, both of which improve a little. This gains our confidence that BP is a possible solution, however, we are looking for more improvement (Figure 15).

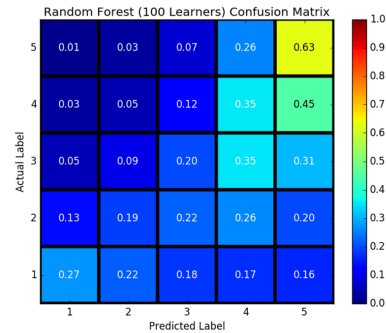


Figure 15. Confusion matrix for fourth round prediction.

Finally, inspired by the idea that attribute "user average stars" gives around 3% increase 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%, 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 for categorization. For example, we separate our user average star and business average star as 0.2 step as one sub

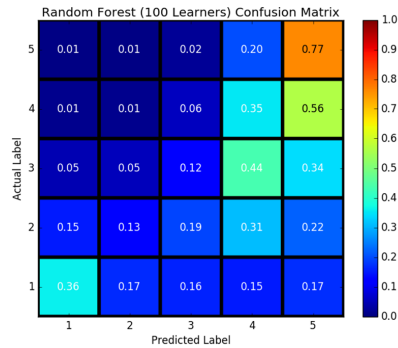


Figure 16. Confusion matrix for fifth round prediction.

category, we will have 21 different levels for both users and restaurants. Then we take into frequent users and frequent restaurants into consideration, taking average score for each user node by its connected restaurant node. We would have at most 5^{21} element 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 results in 640 GB memory requirement which is easy to get 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 text. Our feature vector is based on extracting the sentimental words, calculating related parameters as feature vectors. 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 review. Not only taking out positive/negative words, but also digging more into their relationship. This requires a lot of time into studying human beings emotional expression about something. ~~What is more, we have an expectation that the way how the user describes a restaurant also matters in giving his final rating. Each user has its own habit in writing words, but generally human beings has a trend of describing one time experience from different aspects, for example in restaurant, he might talk about the time he waits for service, whether the food is delicious or not, how is the rate of the price. We have not extracted those promising features out as they might be very important in overall evaluation.~~ Some other possible work would locate in connecting other columns in database into 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 relationship between the location of restaurants and their overall rating level. 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 presented two visualized tools for Yelp users, and after that we extracted multiple feature vectors from clean data and explored different machine learning algorithms to find 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 difficult prediction from extreme users, we import BP method and test on simple case for verification. This gives us promise it would be a better way for improvement. Then we apply it on our data and it shows that after we approximate the data to 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] P. Nathan. Intro to apache spark. <http://stanford.edu/rezab/sparkclass/slides/itas-workshop.pdf>.
- [8] S. Penchikala. Big data processing with apache spark. <http://www.infoq.com/articles/apache-spark-introduction>, 2015.
- [9] 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.

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())
```