MG4E1
Algorithmic Techniques for Data Mining

Mining For Restaurant's Rating: Yelp Case Study

Candidate number
13709
28414
43130

Taught By: László Végh

2015/2016

Department of Management

Contents

Abstract

This report aims to perform a data mining analysis on the business characteristics and text reviews from Yelp's round 7 dataset challenge. 2486 restaurants from Las Vegas are selected as our instances for this research. The research is then divided into two parts. The first part is to explore the relationship between business characters and the ratings of the respective restaurants. The second part is to analyse the keywords from text reviews and its ability to predict the restaurants' ratings. Three types of classifiers – OneR, C4.5 (J48) and Naïve Bayes are applied for the business dataset, while KNN (IBK), Jrip (RIPPER) and Multinomial Naïve Bayes are used for the review dataset. From our experiments, we discover that both datasets are able to provide us with insights on how the stars will change depending on the nature and value of the attributes.

1. Introduction

1.1 Background and Aims

Yelp is one of the most popular and prominent crowdsourcing platform founded in 2005 so that users can review and rate the successfulness of registered local restaurants). The business owners can update their contact information, opening hours and other basic listing information while customers ('Yelpers') can write text reviews and rate the restaurants from 1 to 5 stars, and build a community based on their interactions through reviews. Using Yelp, local restaurants can promote to a lot more potential customers as well as increase their standards based on useful reviews. Besides, customers are able to make use of these information to identify restaurants which suits their needs. Up till 2016, 102 million reviews have been generated using this platform, with up to 90 million unique mobile visitors across more than 33 major cities globally presented in Figure 1 (Statista, 2016).

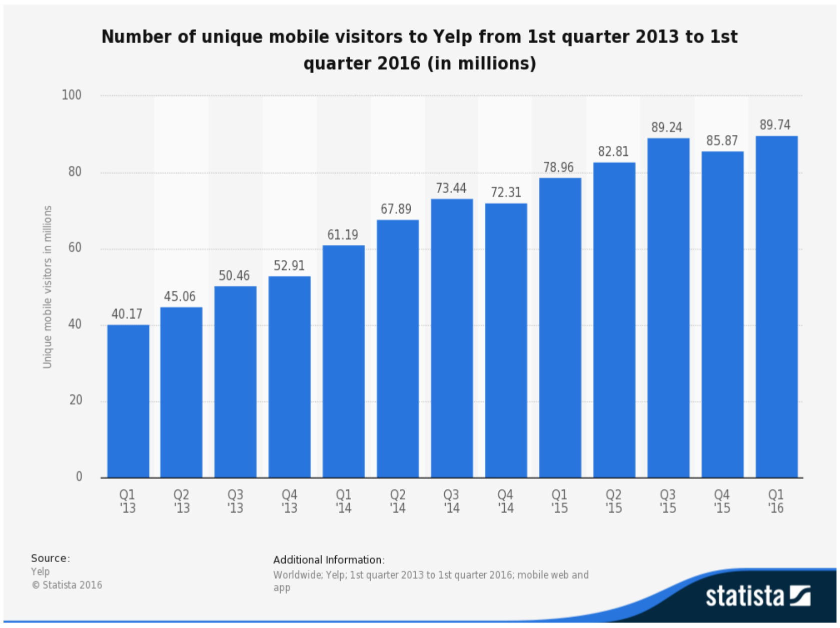


Figure 1. Number of mobile yelpers from 2013 to 2016

After 11 years of operation, "Yelping" has become a common pre-dining habit and this helps Yelp gather extensive amount of valuable raw data regarding different restaurants. According to recent research, its ratings play a significant role in the success of local restaurant as a half star rise will lead to a 19% increase in sell out (Anderson & Magruder, 2012). This suggests these indicator does impact the business to a certain degree. In order to fully explore the functionality of this indicator, we have decided to focus on using the Yelp dataset. We aim to explore questions which include

- Which business attribute attract customers most when they are searching for a place to dine at?
- What is the most frequent words that customers tend to use when they write their reviews?
- How useful is the review dataset when it comes to predicting number or stars achieve?

Considering the complexity of natural language posed in reviews, as well as the significant size of registered businesses and user base; for this research, we aim to mine and summarize business characters and customer reviews of the restaurants in the Yelp database.

The report will be structured as follow: after describing the original dataset information in the later part of Section 1, we will tailor our dataset with preliminary analysis to fit our main research questions in Section 2. Consequently, an explanation of how the data are process will be documented. In Section 3, we will introduce relevant data mining techniques applied to tackle our research objectives with detailed results presented in Section 4. We will then review our project further with possible limitations in Section 5. We will end of this with a conclusion under Section 6.

Throughout this project, we have used idenpython and R programming languages to preprocess initial datasets while applied WEKA to conduct data mining analysis.

1.2 Information about Dataset

The datasets used in this project were obtained from round seven of "Yelp Dataset Challenge" which can be found at:

https://www.yelp.co.uk/dataset_challenge

The website provides 4 datasets in .json file, covering business data, reviews, users and check-in information. However, we will only be using business data and reviews dataset. Our aim is to identify the relationship business data and reviews have on stars rating.

The 380-megabytes "business" dataset contains 77446 business instances with 27 attributes across 10 cities (Edinburgh, Karlsruhe, Montreal, Waterloo, Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, Madison). A full list of attributes is displayed below:

```
'type': 'business',
'business id': (encrypted business id),
'name': (business name),
'neighborhoods': (hood names),
'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
'hours’: {(day of week): (opening hours)}
'price range': ($, $$, $$$, $$$$)
'accepts credit cards': True / False
'waiter service': True / False
'ambience_classy': True / False
'ambience_trendy': True / False
'ambience_romantic': True / False
'ambience_divey': True / False
'ambience_touristy': True / False
'ambience_upscale': True / False
'ambience_intimate': True / False
```

```
'ambience_hipster': True / False
'parking': True / False
'good for': (personalized tag)
```

On the other hand, the 1.2-gigabyte "review" dataset contains 2.2 million text reviews from 552K users and this correspond to all listed instances in business datasets. The features of 7 attributes are presented as follows:

```
'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)}
```

Despite the availability of full-range attributes in both datasets for us to obtain meaningful information, the large . json files comprises of highly unstructured data and this require us to put in a lot of effort to extract and transform them into WEKA recognizable dataset, which was very challenging in the preprocessing stage.

2. Preprocessing and preliminary analysis

2.1 Research Questions

To ensure that the data extraction process is done meaningfully, we will first define our research questions to narrow down the scope of targeting instances so that a thorough analysis could be carried out. The following questions were proposed:

Q1. What are the most important business characters attracting high star rating of the local restaurants?

Q2. What are the most frequent words that appear in the review content and how is it relevant to the restaurant star rating system?

Q3. How can business (restaurant) owners use these analysis to improve their food and service?

Q4. How can new entrants use these information to decide what to look out for?

2.2 Dataset modification

With the guidance of our proposed research questions, we then modify and transform our initial datasets into .csv format. This includes identifying a suitable research city, filtering out missing values and irrelevant attributes, reorganising and regrouping unstructured attributes, and most importantly running preliminary test on text reviews to spot the most frequent words. All these involve using Python and R coding at the pre-processing stage. These data transformation can be found under Appendix1.

2.2.1 Business dataset

The first step was to decide a research city. This is necessary because it does not make sense to compare different cities, and narrowing down on a city will reduce the number of instances which in return allows WEKA to run the analysis more quickly. Since our goal is to explore the business characteristics, an ideal city would be the one that contain the highest instances, with complete (non zero value) set of informative attributes. For this reason, we eliminate all the instances with missing values and select the city with the highest instances remaining. After the elimination process, we observe that:

City name	Number of remaining instances
Edinburgh	982
Karlsruhe	898
Montreal	1197
Waterloo	262
Pittsburgh	1002
Charlotte	1557
Urbana-Champaign	262
Phoenix	1895
Las Vegas	**2486**
Madison	631

Table 1. Yelp business dataset: number of remaining instances in cities

Las Vegas, the city with the highest number of instances was chosen to be the target city. All other instances outside this region would no longer be considered in this research.

We then continue to process the attributes. Considering that we have selected Las Vegas as our single geographical area, any geographical attributes like “city”, “state” seemed to be irrelevant in this content and should be omitted. In fact, in order to discover any meaningful relationships between business characters and star rates, we only select relevant attributes which will allow us find meaningful relationship. Therefore, we have identified certain keywords with the same meaning and combined them. Table 2 below shows the words that we have processed. For example, we aggregated the words “staff”, “service”, “server”, “waiter”, “waitress”, “served” and put them in a column named “service”.

	Before	After
1	“staff”, “service”, “server”, “waiter”, “waitress”, “served”	“service”
2	“price”, “prices”	“price”
3	“priced, overpriced, expensive”	“expensive”
4	“drink”, “drinks”	“drink”
5	“dish”, “dishes”	“dish”
6	“recommend”, “recommended”	“recommend”
7	“star”, “stars”	“star”
8	“sandwich”, “sandwiches”	“sandwich”

9	“friend”, “friends”	“friend”
10	“love”, “loved”	“love”
11	“enjoy”, “enjoyed”	“enjoy”
12	“review”, “reviews”	“review”
13	“burger”, burgers”	“burger”
14	“like, liked”	“like”
15	“dessert”, “desserts”	“dessert”
16	“appetizer”, “appetizers”	“appetizer”
17	“quick”, “quickly”	“quick”
18	“cheap”, “reasonable”	“reasonable”
19	“perfect”, “perfectly”, “perfection”	“perfection”
20	“disappointed”, “disappointing”	“disappointed”

Table 2. Attributes after combining similar keywords

Upon further analysis, the “hours” attribute for most of the restaurants have different combinations of morning opening hours and evening closing hours, from Monday to Friday. This makes it really difficult for us to classify the attribute into a reasonable number of classes to be analysed in the mining process. As a result, we have omitted the “hour” attribute.

Also, both “good for” and “categories” attributes contain personalized or localized tags which are highly unstructured information that make it difficult for us to do mining analysis. The large number of unique

Despite the highly unstructured nature of “categories” attribute, keywords like “Chinese, Thai, Late-American, Pizza” are easy to extract. Therefore, we as human annotator and went through to redefine 2486 instances into 90 general categories. A full list of the categories is provided in the Appendix2. While “good for” attributes are far more complicated in language structure and impossible to annotate and classify so that we will omit “good for” attribute.

Ultimately, we have created a .csv file with the following attributes:

Attribute	**Type**	**Range**	**Description**
Business id	String	N/A	Restaurant unique identity
Review count	Numerical	3-5642	Number of reviews each restaurant receive
Categories	Nominal	1-90	Each number represent one type of restaurant, e.g. 1= Latin American, 3=Pizza. (A full list in Appendix 2)
Price range	Nominal	1-4	1=$ (below $10), 2=$$ ($11-$30), 3=$$$($31-$60), 4=$$$$(above $60)
Accept credit cards	Nominal	0-1	1=True 0=False
Waiter service	Nominal	0-1	1=True 0=False
Ambience – classy	Nominal	0-1	1=True 0=False
Ambience – trendy	Nominal	0-1	1=True 0=False

Ambience – casual	Nominal	0-1	1=True 0=False
Ambience - romantic	Nominal	0-1	1=True 0=False
Ambience - divey	Nominal	0-1	1=True 0=False
Ambience - touristy	Nominal	0-1	1=True 0=False
Ambience - upscale	Nominal	0-1	1=True 0=False
Ambience - intimate	Nominal	0-1	1=True 0=False
Ambience - hipster	Nominal	0-1	1=True 0=False
Parking	Nominal	0-1	1=True 0=False
Stars	**Nominal**	**1-9**	**1=class 3.5** **2=class 3** **3=class 4** **4=class 4.5** **5=class 2.5** **6=class 2** **7=class 5** **8=class 1.5** **9=class 1**

Table 3. Information of final selected attributes

From the table 2, we can see that there are 9 different classes ranging from 1 star to 5 stars. We have also aggregated these classes into “good” and “not so good” class. This allow for more meaningful comparison and will be explained later. The “good” class would contain stars no less then 4 (4, 4.5, 5) and the “not so good” which contains the rest of other star rating (1, 1.5, 2, 2.5, 3, 3.5). The change in class proportions can be visualized below (the rest of final attribute visualization can be found in Appendix3:

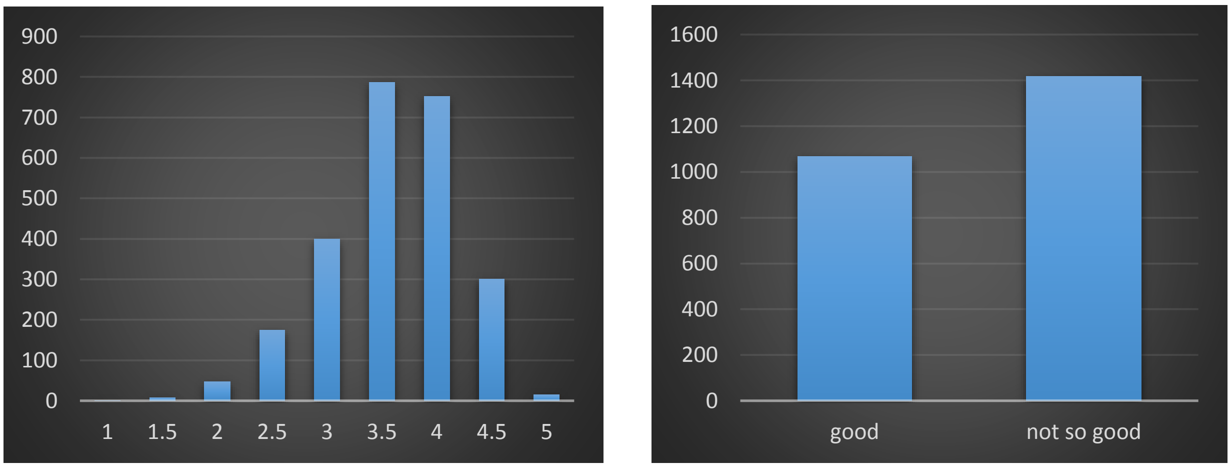


Figure 2. Number of instances in each class before and after aggregating

We will perform data mining analysis on both the datasets and identify which dataset yields better results.

2.2.2 “Review” dataset

The first step of this pre-processing stage was to extract and return the dataset which corresponds to the “business id” under “business dataset”. This process reduced the number of reviews from 2.2 million to 430 thousand which makes our work much more efficient in time. In our second research question, we are only concern about the relationship between review content and star rating. Therefore, we picked “business id”, “text”, “stars” attributes and eliminated attributes “type”, “date”, “user id” and “votes”, which are considered irrelevant in our content.

The second step was to identify the most frequent keywords remaining in the “Review” dataset. We have performed the basic text mining skill using R to find out the word frequency of every unique word (lower and upper cases are not separated) and selected the top 200 frequent words as our keywords of the review content (Appendix 1). These 200 frequent words appear more than 10,000 times in total and is more meaningful when used. Amongst the words that appeared more than 10,000 times, many of these words were “nonsense” words (eg the, I, you) and irrelevant in our research. We have excluded such words in our analysis. Ultimately, we could not simplify the dataset by just picking the top frequent words as keywords but rather need to go through the frequency list manually and determined the top 200 relevant keywords. A summary of selected keywords is presented in the table below:

Positive	**Neutral**	**Negative**
Good, great, best, nice, well, delicious, pretty, better, amazing, fresh, friendly, free, high, worth, sure, awesome, favourite, excellent, big, new, sweet, tasty, happy, special, huge, super, decent, clean, fun, fantastic, yummy, crispy wonderful, recommended, flavorful, impressed, expected, prime, beautiful, wow, juicy, share comfortable, cute, interesting, incredible, outstanding, yum, recommend, love, friend, enjoy, like, quick, reasonable, perfect, loved, enjoyed, liked, quickly, cheap, perfectly, perfection	Food, so, very, back, really, only, chicken, menu, definitely, sauce, always, cheese, dinner, everything, bar, pizza, experience, steak, salad, fries, food, sushi, lunch, side, most, meat, hot, beef, both, flavor, fried, bread, quality, rice, shrimp, soup, pork, crab, ok, hotel, location, spicy, hour, atmosphere, fish, chocolate, maybe, cream, wine, beer, lobster, location, bacon, water, party, coffee, seafood, thai, tacos, friends, French, absolutely, rib, paste, salmon, tea, size, deal, start, cake, grilled, butter, Mexican, tuna, noodles, music, Italian, Chinese, hungry, reviews, ambience, curry, Asian, pieces, vodka, Japanese, service, price, drink, friend, drinks, dish, dishes, star, seated sandwich, review, burger, dessert, appetizer, buffet, boyfriend, staff, waiter, waitress, served, eating, sandwiches, burgers, desserts, appetizers, seating, anything, yelp, garlic, eggs, casino, priced, overpriced, noodles, stars, onion, prices	Busy, wrong, never, bad, small, waited, slow, horrible, dirty, crowded, terrible, expensive, disappointed, disappointing

Table 4. Summary of keywords

We can see that majority of the keywords are neutral words like chicken, cheese, pizza which covers the food topics. A third of the keywords are positive words while only 7% of the keywords are negative words. This shows that reviewers tend to use words having positive polarity more than words with negative polarity.

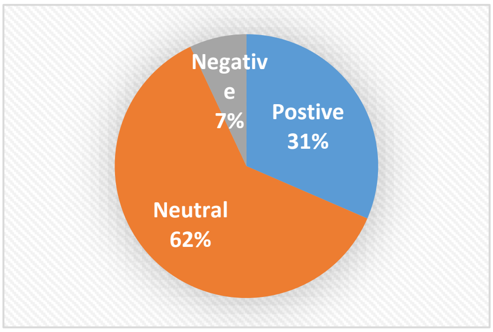


Figure 3. Distribution of polarity of keywords

The third and the most important step was to find out the frequency of defined keywords in the collective reviews of individual restaurant. This process involves transforming the string characters into numerical attributes with its respective count, allowing us to explore the possible superficial relationship between keywords and local restaurants' star rating system in the data mining analysis. The frequency results were then presented in a table where each keyword is treated as attribute. The "star" attribute on the right is the class which we will be trying to predict. Table 5 below provides an example of our sorted data. For example, the first column (business id cvJDHk1ho0DxaF26etcr8Q) has the word "good" appearing 53 times after considering ALL the review of this restaurant.

Business id	**good**	**fresh**	**service**	**...**	**star**
cvJDHk1ho0DxaF26etcr8Q	53	7	62	...	3.5
t0PNqBiITGf_AY6uNsaD5g	11	0	3	...	3.5
8izNisB-aS1dOB6cFVZdPg	10	5	8	...	3.5
YZxA6w82eFJFntwqJLln6w	2	0	0	...	3
...	...	...	...	...	...

Table 5. Sum of count of each individual word for each restaurant

After reviewed all the attributes in the table, we realized that some of the keywords have the same meaning expressed in different terms which is a common issue when handling natural language text. Therefore, regrouping attributes with the same meaning attributes should be done. This includes "staff, service, server, waiter, waitress, served", "price, prices", "priced, overpriced, expensive", "drink, drinks", "dish, dishes", "recommend, recommended", "star, stars", "sandwich, sandwiches", "friend, friends", "love, loved", "enjoy, enjoyed", "review, reviews", "burger, burgers", "like, liked", "dessert, desserts", "appetizer, appetizers", "quick, quickly", "cheap, reasonable", "perfect, perfectly, perfection" and "disappointed, disappointing" (20 pairs). The whole merge process reduced the number of attributes from 202 (including business id and star attribute) to 172.

Finally, the review dataset with sorted attributes in .csv file was ready for us to do further investigation. However, our team realized that an inherent different in word count between restaurants will occur. This is because larger restaurants which have been registered with Yelp for a longer period of time will naturally have more reviews, and a smaller restaurant will have lesser reviews in total. As a result, the former category will have higher individual word count compared to the latter.

Therefore, in order to make this review more comparable, we have divided the number of sum of count of each word for each restaurant, by the number of review the restaurant has.

2.3 Attribute selection

Given the large number of attributes, our team wants to narrow down and predict the class(outcome) based on certain attributes which are more significant and has more predictive strength, than using generic ones with less predictive power. Hence, we have used infogain attribute evaluation to evaluate the worth of an attribute. WEKA will measure the information gain with respect to the different classes. Also, we have use chi squared attribute evaluate, where this feature measure each chi squared statistic, with respect to each class.

INFOGAIN	**CHI SQUARED**
0.1507 14 delicious	547.58742 14 delicious
0.1353 50 favorite	467.23494 159 love
0.1323 17 amazing	445.87848 50 favorite
0.1281 159 love	442.11818 17 amazing
0.1212 169 perfect	421.10601 169 perfect
0.116 21 definitely	416.71048 21 definitely
0.1132 95 fantastic	391.26662 49 awesome
0.1126 149 yum	385.67558 149 yum
0.1096 49 awesome	380.02709 10 best
0.1072 98 yummy	378.16715 6 great
0.1057 47 flavor	366.52567 98 yummy
0.1055 103 wonderful	364.5712 95 fantastic
0.1045 6 great	363.5196 47 flavor
0.1022 10 best	345.16776 103 wonderful
0.1019 52 excellent	330.78776 52 excellent
0.0981 34 friendly	330.4529 34 friendly
0.0967 119 flavorful	319.91438 155 recommend
0.0959 18 fresh	314.26392 18 fresh
0.0943 144 incredible	308.97798 119 flavorful
0.0936 155 recommend	296.38639 144 incredible

Table 6. Attributes selected using the respective methods

From the list above, the attributes are listed according to their ranks and we can see that delicious is the best attribute. Coincidentally, the 20 attributes returned using the 2 evaluation methods are the same but in different order. Therefore, we team will proceed with the prediction of the stars using these 20 attributes.

3. Data mining methods

After finishing the data preprocessing, this section will introduce the evaluation methods and classification learning algorithms applied in Weka.

3.1 Evaluation methods

Evaluation methods such as confusion matrix, kappa statistics and correlation coefficient are used to measure and compare the performance of different methods.

Confusion Matrix

The confusion matrix is represented in the table below, and it is used to display the result of the classifier on a test data given the actual values that are already known (Stehman, 1997).

		Actual	
Predicted		Yes	No
	Yes	True positive (TP)	False positive (FP)
	No	False negative (FN)	True negative (TN)

Table 7. Confusion matrix

There are several levels of rating stars for local restaurants, and we would take rating star class2 as an example. True positive indicates that those star-class2 restaurants are correctly predicted as star-class2. Similarly, true negative also correctly identified restaurants that are not star-class2 in the corresponding classes. False negative represents those star-class2 restaurants are misclassified in other classes, while false positive means that restaurants belong to other classes are not correctly identified in the class2. With the concepts from confusion matrix, the error rate, precision and recall rate for each method could be formulated.

Precision & Recall

As for these rates, we aim to maximise their values. The precision is the fraction of all correctly identified instances among positive predictions, and could be calculated by *TP / (TP+FP)*. Recall is the fraction of positive instances identified and could be calculated by *TP / (TP+FN).*

Kappa Statistics

Another essential evaluation method is kappa statistics, which compares our classifier with a random guess to measure the performance of the methods. The formula is represented with the use of Diagonal (D):

Kappa statistics = [D (observed) – D (random)] / [D (perfect) – D (random)]

Where the D (observed) is the total number of successfully identified instances in our current classifiers, and D (perfect) stands for the sum of instances (Witten & Fran, 2011). The higher value of kappa statistics, the better the classifier will be.

Correlation Coefficient

It is a quantitative measure which used to indicate the degree of linear dependencies and statistical relationships between variables (Witten & Fran, 2011). The value range is between -1 and 1, where -1 shows a negative correlation while 1 represents a totally positive correlation.

3.2 Classification methods

Linear Regression is a statistical method to model the relationship between one dependent variable and one or more independent variables (Witten & Fran, 2011). The approach adopted in our project is called multiple linear regression where the dependent variable is the rating-star and many independent variables (e.g., review count, parking).

OneR rule is a simple and basic classifier which generates one rule and gives the prediction based on single attribute. In order to construct a rule as predictor, the frequency table would be used to compare the accuracy of different attributes and select the one with lowest total error (Witten & Fran, 2011). This method is always useful for benchmarking and prediction.

Naïve Bayes rule is a simple probabilistic classifier based on Bayes' theorem assuming all attributes are independent from each other which are not true in reality. The probabilities of classification will be computed according to the equation presented below:

$$Pr[H \mid E] = \frac{Pr[E \mid H]Pr[H]}{Pr[E]}$$

Pr [H|E] is the conditional probability of one hypothesis H while assuming that evidence E holds. Similarly, Pr [E|H] represents the conditional probability of Evidence under the hypothesis H. The likelihood of the class could be regarded as the H, and evidence E is assigned to represent the instances to be classified. According to the independence assumption, the probability could be calculated in this way:

$$Pr[E \mid H] = Pr[E_1 \mid H] \cdot Pr[E_2 \mid H] \cdot Pr[E_3 \mid H] \cdot Pr[E_4 \mid H]$$

$$Pr[H \mid E] = \frac{Pr[E_1 \mid H] \cdot Pr[E_2 \mid H] \cdot Pr[E_3 \mid H] \cdot Pr[E_4 \mid H] \cdot Pr[H]}{\Pr[E]}$$

Besides, this method would omit the corresponding attributes when encountering missing values and adopt the Laplace estimator for handling zero frequency by adding one for each class (Witten & Fran, 2011). Another important method relating to Bayes' theorem is called **Multinomial Naïve Bayes**, and the formula is shown as follows:

$$\Pr[E|H] = N! \cdot \prod_{i=1}^{k} \frac{P_i^{n_i}}{n_i!}$$

It could be regarded as a specialised version of Naïve Bayes for text classification. P (i) shows the probability for k possible words and every word is selected based on these probabilities. And *nk* indicates the frequency of k words.

K-Nearest Neighbours Algorithm (IBK in Weka) is a type of instance-based learning. The output of KNN is to classify the instance, and distance measurement will be applied to compare and the object will be classified by the majority of its neighbours. For instance, if k is 1, then the instance will be assigned to the single class. When implementing this algorithm in Weka, k = 2 and 9 are used for the dataset.

C4.5 is the extended version of ID3 algorithm, which is used to generate the decision tree for classification and we use J48 in Weka for generating the pruned or unpruned trees. As an extended algorithm based on the 'Divide-and-conquer', we can handle the training set with unknown attributes values through computing and comparing the information gain. It performs the reduced error pruning and post-pruning rule, and could handle continuous attributes (e.g., temperature). Decision trees would be easier for interpretation and explanation comparing to other methods, and the minimum number of instance was set to 200 for simplifying the tree when implementing this method in Weka.

RIPPER (JRip in Weka), an abbreviation for Repeated Incremental Pruning to Produce Error Reduction, which is a propositional rule learner and covering classifier (Witten & Fran, 2011). It is a sophisticated method and using entropy gain measure for rule construction. Rules that include most of the objective class are generated from each iteration, and those instances covered by the former produced rules will be removed for preventing for the next iteration. The iteration will not stop until these generated rules could cover the target class only. Its accuracy is improved by such constant changing and replacing rules and the pruning steps would prevent overfitting. When implementing in Weka, all parameters are kept in the default setting.

4. Evaluation of different methods

As seen from the tables below, we can see the results are generally ineffective (low hit rate, precision, recall and kappa statistic). All classifier test a sample size of 2486 instances.

Predicting stars based on text attributes selected					
Classifier	*Correctly classified instances (Hit rate)*	*Precision*	*Recall*	*Kappa*	*Correlation Coefficient*
9-NN	46.50%	0 - 0.453	0 - 0.625	0.2739	
Multinomial Naive Bayes	34.19%	0 - 0.539	0 - 0.948	0.0386	
J-RIP	39.90%	0 - 0.472	0 - 0.705	0.1635	
Linear Regression					0.6211
Predicting stars based on business data					
Classifier	*Correctly classified instances (Hit rate)*	*Precision*	*Recall*	*Kappa*	*Correlation Coefficient*
1R	31.17%	0 - 0.453	0 - 0.554	0.0212	
J48	35.00 %	0 - 0.325	0 – 0.644	0.0561	
Naive Bayes	32.82%	0 - 0.359	0 - 0.69	0.0937	
Linear Regression					0.2276

Table 8. Results collected after running data using 9 different classes

Before sorting the classes into Good and Not so good, there are 9 classes which we are trying to predict, namely 1 star, 1.5 stars, 2 stars, 2.5 stars, 3 stars, 3.5 stars, 4 stars, 4.5 stars and 5 stars. Logically, the larger number of classes will result in higher error because there is a lower margin for error, where the instances have to be either correctly classified or wrongly classified.

For predictions based on text attributes selected with 9 classes of prediction, K-NN is the best method with about 42% of instances classified correctly. Precision and recall takes a range or values as seen in Table 8 above. Kappa statistic is highest when K-NN is used, with a value of 0.2739 which is considerably low. The precision and recall values are higher for 3 – 4 stars. Refer to Appendix for the values.

Since the selected text are positive words, such as favorite, delicious, perfect and wonderful, reviews of businesses with low star count might not contain such reviews since a low star count suggest that the reviewer does not have a good experience and would be writing reviews with negative words such as dislike, tasteless, noisy and unappetizing. We believe that is why precision and recall for low star businesses tend to be low as well and tend to take the value 0.

Looking at the prediction of stars based on business data, we can see that the correctly classified instances lies around 31 to 33 percent. For the 1R rule, it returns "Business ID" as the best attribute for predicting the number of stars. This is clearly a case of over fitting since every instances has its own business id. Our team therefore remove this attribute and ran the 1R rule again and the resulting attribute used for prediction then became review count. The results of how the stars awarded changes according to the number of reviews can be seen in Appendix. What is interesting here is that the prediction made are mixed between 3, 3.5 and 4 stars and as the review count goes higher than 145, the remaining predictions are of 3.5 and 4 stars. This shows that as review count for a business goes beyond a certain point, the number of stars generally increases. It should be noted that 1R rule is normally used as a benchmark when analysing predicted results.

Using Naïve Bayes as the classifier, the Kappa statistic returns the highest value – 0.0937. What is important to take note here is that Kappa statistic and the correctly classified instances of all the classifier used are very low and the results are ineffective should it be used to predict future stars based on business data provided. This is a results of how the data has been manipulated we can see that using the results above might be less effective when business try to improve their operations based on information provided.

It is therefore more meaningful to analyse results after stars are sorted into 2 classes, and also, it is more accurate for business owners to see where they can improve on to increase their stars rating. This also make practical sense since all business owners are looking to improve their business services and push their restaurant into the "Good Restaurant category". For this report, our team will be focusing more on the data after the information have been sorted into 2 classes, namely "Good" and "Not so good".

Predicting stars based on text attributes selected (stars sorted into 2 classes)							
Classifier	*Correctly classified instances (Hit rate)*	*Precicion (Good)*	*Precicion (Not so good)*	*Recall (Good)*	*Recall (Not so good)*	*Kappa*	*Correlation Coefficient*
2-NN	72.12%	0.809	0.693	0.46	0.918	0.3992	
Multinomial naive bayes	57.04%	0	0.57	0	1	0	
J-RIP	75.10%	0.73	0.765	0.668	0.814	0.4865	
Linear Regression							0.5188
Predicting stars based on business data (stars sorted into 2 classes)							
Classifier	*Correctly classified instances (Hit rate)*	*Precicion (Good)*	*Precicion (Not so good)*	*Recall (Good)*	*Recall (Not so good)*	*Kappa*	*Correlation Coefficient*
1R	58.13%	0.519	0.607	0.349	0.756	0.1097	
J48	60.74%	0.563	0.626	0.385	0.775	0.1664	
Naive bayes	62.27%	0.61	0.627	0.838	0.337	0.1851	
Linear Regression							0.1954

Table 9. Results collected after running data using 2 different classes

The results of after using the different classifiers are shown in Table 9 above.
After sorting the classes into Good (4 – 5 stars) and Not so good (1 – 3.5 stars), we can see the results have significantly improved. Similarly in this case, all classifier test a sample size of 2486 instances.

First, let us take a look at the predicted results after running the select text attributes on WEKA. We can see that K-NN and J-Rip are more effective in providing the relationship between the text selected and the number of stars that a restaurant will receive should the reviews contain the attributes selected. We have selected K to take a value of 2 since there are 2 classes which are interested to predict. It can be insightful to run and collect values using different values of K but our team has decided to let K take value 2 only. This is to make more meaningful comparison against other classifiers.

2-NN classifier gives a precision of 0.693 for "Not so good" businesses and 0.809 for "Good" businesses. Kappa statistic is 0.3992 and this is significantly higher as compared to the results before the classes were sorted. Using J-RIP as the classifier, we have came up with certain rules which can help business identify which areas to work on so that they can improve their stars count. Some of the rules are
(delicious >= 0.160714) and (amazing >= 0.104693) => stars=Good (673.0/136.0)
(amazing >= 0.074074) and (friendly >= 0.133588) and (fresh >= 0.173913) => stars=Good (129.0/44.0)

Looking at the second rule, we can interpret the results as; when the word "amazing" appears an average of 0.074 times per article, the word "friendly" appears on an average of 0.1336 times per review, and as the word fresh appears on an average of 0.17393 per review, the business will be classified under a "Good". Although it might be debatable as to how the word "amazing" should be interpreted, since it can be amazing food, amazing staff or amazing garnish, we can confidently say that having friendly staff and fresh food will lead to the business receiving high number of stars.

Multinomial Naïve Bayes is not so effective as compared to the 2 methods mentioned above, with 57.04% correctly classified instances and a Kappa statistic of 0. To predict if the class of the business based on business data, we have used similar classifier as before – 1R, J48 and Naïve Bayes.

We removed the "Business ID" attribute before running the 1R rule. The rule returned "Review_Count" as the best attribute. Amongst the 3 classifier used, 1R rule returns the lowest precision and Kappa statistic. Using J48 as the classifier and setting the minimum number of instances in each leaf to 200, the tree produced can be seen below.

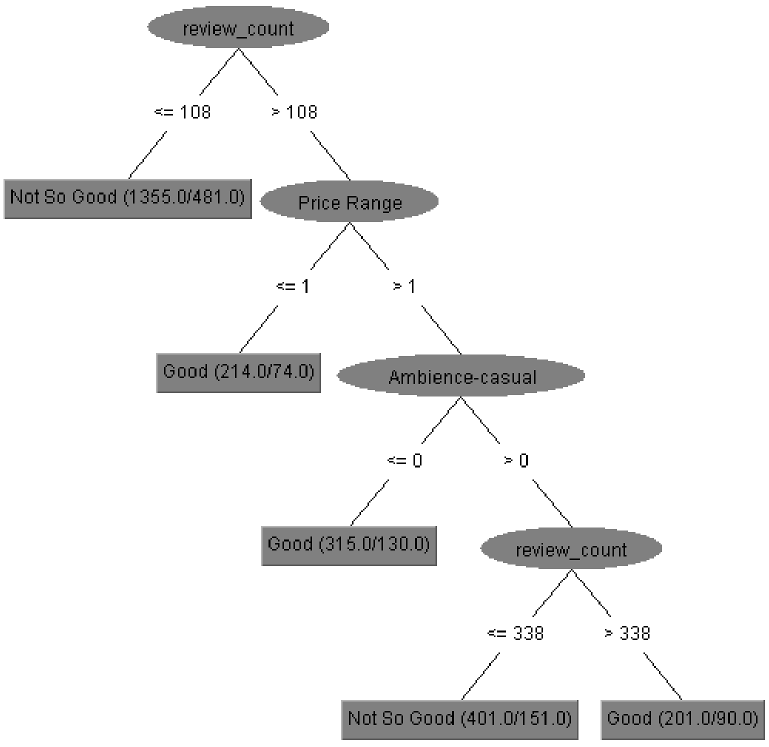


Figure 4. Tree produced using J48

J48 has a hit rate of 60.74%, Kappa statistic of 0.1664. This is higher than the correctly classified rate and Kappa statistic compared in 1R. From the tree constructed, we can see that for the good class, the rules are as shown below

Review_Count > 108 and Price Range <= 1: Good (214.0/74.0)
Price Range > 1 and Ambience-casual <= 0: Good (315.0/130.0)
Review_Count > 338: Good (201.0/90.0)

The tree shows that review count has high importance and as review count increase, the class generally falls into the “Good” class.

On the other hand, it is quite interesting to see that when Price Range is more than one, and ambience casual is False (Ambience is non casual), the business will tend to be in the “Good” class as well. Relating it to real life, new market entrance going into the food industry can consider setting up a business which sells pricier food and does not have a non-casual setting because they tend to receive good reviews and there might be more such demand in future. Ultimately, Naïve Bayes gives the results, correctly classified instances – 62%, precision and Recall for Not so good and Good classes are 0.627, 0.61, 0.337 and 0.838 respectively. Kappa statistic for this classifier is 0.1851. Using Naïve Bayes, the classifier multiplies the probability taking into account all attributes. This however might pose some limitations since the number of attributes increases (current business will have to look into all the different attributes before they can see how to increase move their restaurants into a good class).

All in all, our team end of the analysis by constructing several linear regression to see which attributes will lead to higher stars. Different attributes are given different weights in order to predict the number of stars attained.

Predicting the number of stars (for 9 different classes), the linear equation generated are

stars = 0.1967 * great + 0.3454 * best + 1.6843 * delicious + 1.4815 * amazing + 0.3295 * fresh + 0.4197 * definitely + 0.6034 * friendly - 0.3844 * flavor + 1.0318 * awesome + 0.8935 * favorite – 0.622 * fantastic + 0.7689 * flavorful - 1.2457 * incredible + 0.3017 * love + 2.7446

based on text attributes selected and,

stars = 0.0002 * Review_Count -0.0675 * Price Range + 0.3529 * Ambience-classy + 0.1876 * Ambience-trendy + 0.0687 * Ambience-casual + 0.2555 * Ambience-romantic + 0.2778 * Ambience-divey - 0.4244 * Ambience-touristy + 0.1417 * Ambience-upscale + 0.2732 * Ambience-intimate + 0.1414 * Ambience-hipster + 0.1675 * PARKING + 3.4297

based on business data.

Predicting the number of stars (after sorting data into 2 different classes) gives linear regression model

stars = 0.1871 * best + 1.1401 * delicious + 1.3187 * amazing + 0.2186 * fresh + 0.3153 * definitely + 0.492 * friendly - 0.1945 * flavor + 0.651 * awesome + 0.7322 * favorite - 0.3479 * excellent + 0.8227 * wonderful - 0.1034

based on text attributes selected and,

stars = 0.0002 * Review_Count - 0.0333 * Price Range - 0.0487 * Waiter Service + 0.2583 * Ambience-classy + 0.1989 * Ambience-romantic + 0.1606 * Ambience-divey - 0.3616 * Ambience-touristy + 0.1747 * Ambience-intimate + 0.0716 * PARKING + 0.3917
based on business data.

The correlation coefficient for the 4 models are 0.6211, 0.2276, 0.5188 and 0.1954 respectively.

In this case, it more meaningful to look at data before the classes are sorted/combined into 2 classes because users are able to calculate the actual number of stars a certain business will get when different factors are accounted for. This is especially true when predicting the number of stars (for 9 different classes) because the coefficient correlation is 0.6211, where the strength of the linear relationship is higher and using this equation to predict the number of stars give a more accurate result.

5. Limitations

During the preprocessing stage, we narrowed down the number of total instances by selecting Las Vegas as the only object of study and also omitted those restaurants with missing values. This could be a biased process if take the Yelp Dataset as a whole, as the data based on single city are not representative for all and customers from different regions and countries have various perceptions about among the restaurants. Thus, it is essential to concentrate on other the rest data as well for the future research.

As for those ‘business’ data, we also manually filtered some attributes during preprocessing. And for the ‘review’ part of dataset, it is worthy to make good use of the text mining technique for exploring the in-depth research field. With the limit of knowledge on this area and inadequate skills about natural language process, our techniques applied for analysis would narrow the research field on this dataset and the result may not perform well than expected. Whilst trying to select the best attribute to be used for this analysis, our team used Infogain as well and Chi-Squared method to select the best 20 words as mentioned at the start of the report. The problem with this is that most of the word chosen are mainly positive words (delicious, yummy, fresh, tasty) which suggest that reviewers tend to focus on the quality of food. Although these words are meaningful, it does not provide much insight to business owners as how to now they can improve to increase their stars rating. This is because it is understood that customers are value good food but such review provide business owners with little useful information.

The individual adjective or noun can be rather misleading since we are unable to interpret some of the words to the objects which they are linked to. In this case, positive words does lead to higher stars rating and it might be safe to assume that negative words will lead to lower stars rating, where the opposite applies. However, we cannot confirm that is true since that analysis has not been conducted.

In order to provide more meaningful analysis, perhaps group of words with an adjective followed by a noun because the study of single frequent word cannot capture customers’ reviews whole meaning and may lead to biased research result. Some examples are “big table”, “spacious room” or “comfortable chairs”. This way, we are able to pin point certain unambiguous areas which business owners can work on to improve their service and quality and money can be channel into these areas should the business plan to expand.

6. Conclusion

Ultimately, even though our team has faced certain limitations and restrictions during the course of this analysis, we find that the report to a certain extent, has been successful in answering the questions identified.

When it comes to business characteristics, customers tend to focus on the price and how casual (ambience) the restaurant tend to be. Therefore, restaurant owners should focus on providing food which are value for money because this is what reviews prioritise. Interesting, as the number of review increases, the number of stars tend to go up. Therefore restaurant owners should entice patrons to write review for them on Yelp. Restaurants can perhaps provide discounts for reviewers if they write a review on the spot after having a meal in order to increase their review count. Our team believes that the there is a larger pool of reviewers who writes positive reviews – offering compliments compared to the number of reviewers writing complain, and that is why as the number of reviews increase, the number of stars increase in the long run.

On the other hand, after conducting certain text analysis, our team has identified that as keywords like "delicious, amazing, friendly and fresh" appear more frequently (based on the rules constructed), the number of stars increased as well. As mentioned earlier, since customer value friendly staff, existing restaurants/new entrants should focus on training and upgrading the skillset of their waiter and waitress so that they can provide better services and make their experience at the restaurant more enjoyable.

This is particularly important for new entrants who are looking at entering the food industry. They should pay particular attention to the needs of the customers as mentioned above and "set the tone right" from the beginning.

Reference

Anderson, M. & Magruder, J. (2012). Learning from the Crowd: Regression Discontinuity Estimates of the Effects of an Online Review Database. *Economic Journal*. 122 (563), 957-989. Available from: http://econpapers.repec.org/article/ecjeconjl/v_3a122_3ay_3a2012_3ai_3a563_3ap_3a957-989.htm [Accessed 18th May 2016]

Statista. (2016). Yelp: number of unique mobile visitors 2013-2016. Available from: http://www.statista.com/statistics/385440/unique-mobile-visitors-yelp/ [Accessed 19th May 2016]

Steham, S.V. (1997). Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment.* 62 (1), 77-89. Available from: http://www.sciencedirect.com/science/article/pii/S0034425797000837 [Accessed 19th May 2016]

Witten, I. H., & Frank, E. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). Morgan Kaufmann.

Appendix

Appendix 1 Python and R codes for preprocessing

Python

```
# coding:utf-8
import csv

def GetCSVContent(csvpath):
    ContentData = []
    try:
        reader = csv.reader(file(csvpath, 'rb'))
        ContentData = [x for index, x in enumerate(reader) if index > 0]
    except Exception, e:
        print e
        print "input file not exist or already opened!"
    return ContentData

def WriteCSVContent(csvpath, header, content, encoding="GB18030"):
    try:
        writer = csv.writer(file(csvpath, 'wb+'))
        writer.writerow(header)
        for row in content:
            writer.writerow(row)
    except Exception, e:
        print e

if __name__ == '__main__':
    header = ["中国", "To", "Transfered", "balance", "State"]
    content = [['Dravosburg', '4', 'Mr Hoagie', '', 'True', '5UmKMjUEUNdYWqANhGckJw', '-79.9007057', '21:00',
'11:00', '21:00', '11:00', '21:00', '11:00', '21:00', '11:00', '21:00', '11:00', 'PA', '4734 Lebanon Church
Rd\nDravosburg, PA 15034', '4.5', '40.3543266', '11:00', '11:00', '11:00', '11:00', '11:00',
        '11:00', '11:00', '11:00', '11:00', '11:00', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False',
'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False', 'False',
'False', 'False', 'business', u'Fast Food,Restaurants']]
    WriteCSVContent('test_out.csv', header, content)

import FileHandler
import json
import types
import copy
import sys
import re
reload(sys)
sys.setdefaultencoding('utf-8')


def converttocsv(infilename, outfilename):
    result = []
    # headers = []
    headers = [u'city', u'review_count', u'name', u'neighborhoods', u'type', u'business_id', u'full_address',
            u'hours', u'state', u'longitude', u'stars', u'latitude', u'attributes', u'open', u'categories']
    headers_new = []
    with open(infilename, 'r') as f:
        for line in f:
```

```
        obj = json.loads(line)
        newrow = []
        for title in headers:
            valuetype = type(obj[title])
            if valuetype in [types.UnicodeType, types.IntType, types.BooleanType, types.FloatType]:
                newrow.append(str(obj[title]))
                headers_new.append(title)
            if valuetype is types.ListType:
                newrow.append(",".join(obj[title]))
                headers_new.append(title)
            if title == "hours":
                for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]:
                    if obj[title].has_key(day):
                        newrow.append(str(obj[title][day]["open"]))
                        newrow.append(str(obj[title][day]["close"]))
                    else:
                        newrow.append("")
                        newrow.append("")
            if title == "attributes":
                for attr in ["Outdoor Seating", "Attire", "Price Range", "Takes Reservations", "Caters", "Good for
Kids", "Delivery", "Accepts Credit Cards", "Alcohol", "Good For Groups", "Noise Level", "Drive-Thru", "Waiter
Service", "Take-out", "Has TV"]:
                    if obj[title].has_key(attr):
                        # headers_new.append(title + "-" + attr)
                        newrow.append(str(obj[title][attr]))
                    else:
                        newrow.append("")

                for attr in ["Ambience"]:
                    if not obj[title].has_key(attr):
                        for i in range(9):
                            newrow.append("")
                        break
                    for sub_attr in ["classy", "trendy", "casual", "romantic", "divey", "touristy", "upscale", "intimate",
"hipster"]:

                        if obj[title][attr].has_key(sub_attr):
                            # headers_new.append(
                                # title + "-" + attr + "-" + sub_attr)
                            newrow.append(obj[title][attr][sub_attr])
                        else:
                            newrow.append("")

                for attr in ["Parking"]:
                    if not obj[title].has_key(attr):
                        for i in range(5):
                            newrow.append("")
                        break
                    for sub_attr in ["garage", "street", "validated", "lot", "valet"]:
                        if obj[title][attr].has_key(sub_attr):
                            # headers_new.append(
                                # title + "-" + attr + "-" + sub_attr)
                            newrow.append(obj[title][attr][sub_attr])
                        else:
                            newrow.append("")

                for attr in ["Good For"]:
                    if not obj[title].has_key(attr):
                        for i in range(6):
                            newrow.append("")
                        break
                    for sub_attr in ["brunch", "dessert", "lunch", "dinner", "breakfast", "latenight"]:
                        if obj[title][attr].has_key(sub_attr):
                            # headers_new.append(
                                # title + "-" + attr + "-" + sub_attr)
                            newrow.append(obj[title][attr][sub_attr])
                        else:
                            newrow.append("")

                    # headers_new.append(title)
                    # for (key, value) in obj.items():
                    # print key
                    # headers.append(key)
                    # valuetype = type(value)
                    # if valuetype is types.UnicodeType \
                    #         or valuetype is types.IntType\
                    #         or valuetype is types.BooleanType\
                    #         or valuetype is types.FloatType:
                    #     # print(key, value)
                    #     newrow.append(str(value))
                    #     headers.append(key)
                    # else:
                    #     if valuetype is types.ListType:
                    #         newrow.append(",".join(value))
                    #         headers.append(key)
                    # print(key, ",".join(value))
                    # print valuetype
                    # else:
                    #     if valuetype is types.DictType:
                    #         for subkey, subvalue in value.items():
                    #             if type(subvalue) is types.DictType:
                    #                 for subsubkey, subsubvalue in subvalue.items():
                    #                     fullkey = key + "-" + subkey + "-" + subsubkey
                    #                     # print(fullkey, subsubvalue)
                    #                     newrow.append(
                    #                         str(subsubvalue))
                    #             else:
                    #                 # print(key + "-" + subkey, str(subsubvalue))
                    #                 newrow.append(
                    #                     str(subsubvalue))

                    #     else:
                    #         print valuetype
                    # print len(newrow)
        result.append(newrow)
# headers = [u'city', u'review_count', u'name', u'neighborhoods', u'type',
# u'business_id', u'full_address', u'hours-Tuesday-close',
# u'hours-Tuesday-open', u'hours-Friday-close', u'hours-Friday-open',
# u'hours-Monday-close', u'hours-Monday-open', u'hours-Wednesday-close',
# u'hours-Wednesday-open', u'hours-Thursday-close',
# u'hours-Thursday-open', u'state', u'longitude', u'stars', u'latitude',
# u'attributes-Take-out', u'attributes-Drive-Thru', u'attributes-Outdoor
# Seating', u'attributes-Caters', u'attributes-Noise Level',
# u'attributes-Parking-garage', u'attributes-Parking-street',
    # u'attributes-Parking-validated', u'attributes-Parking-lot',
    # u'attributes-Parking-valet', u'attributes-Delivery',
    # u'attributes-Attire',
        # u'attributes-Has TV', u'attributes-Price Range', u'attributes-Good For-dessert', u'attributes-Good For-
latenight', u'attributes-Good For-lunch', u'attributes-Good For-dinner', u'attributes-Good For-breakfast',
u'attributes-Good For-brunch', u'attributes-Takes Reservations', u'attributes-Ambience-romantic', u'attributes-
Ambience-intimate', u'attributes-Ambience-classy', u'attributes-Ambience-hipster', u'attributes-Ambience-
divey', u'attributes-Ambience-touristy', u'attributes-Ambience-trendy', u'attributes-Ambience-upscale',
u'attributes-Ambience-casual', u'attributes-Waiter Service', u'attributes-Accepts Credit Cards', u'attributes-
Good for Kids', u'attributes-Good For Groups', u'attributes-Alcohol', u'open', u'categories']
    # headers =
    # headers = [u'city', u'review_count', u'name', u'neighborhoods', u'type', u'business_id', u'full_address',
        # u'hours', u'state', u'longitude', u'stars', u'latitude',
        # u'attributes', u'open', u'categories']

    headers_new = [u'city', u'review_count', u'name', u'neighborhoods', u'type', u'business_id', u'full_address',
u'hours-Monday-open', u'hours-Monday-close', u'hours-Tuesday-open', u'hours-Tuesday-close', u'hours-
Wednesday-open', u'hours-Wednesday-close', u'hours-Thursday-open', u'hours-Thursday-close', u'hours-
Friday-open', u'hours-Friday-close', u'state', u'longitude', u'stars', u'latitude', u'attributes-Outdoor Seating',
u'attributes-Attire', u'attributes-Price Range', u'attributes-Takes Reservations', u'attributes-Caters',
u'attributes-Good for Kids', u'attributes-Delivery', u'attributes-Accepts Credit Cards', u'attributes-Alcohol',
u'attributes-Good For Groups', u'attributes-Noise Level', u'attributes-Drive-Thru',
            u'attributes-Waiter Service', u'attributes-Take-out', u'attributes-Has TV', u'attributes-Ambience-
classy', u'attributes-Ambience-trendy', u'attributes-Ambience-casual', u'attributes-Ambience-romantic',
u'attributes-Ambience-divey', u'attributes-Ambience-touristy', u'attributes-Ambience-upscale', u'attributes-
Ambience-intimate', u'attributes-Ambience-hipster', u'attributes-Parking-garage', u'attributes-Parking-street',
u'attributes-Parking-validated', u'attributes-Parking-lot', u'attributes-Parking-valet', u'attributes-Good For-
brunch', u'attributes-Good For-dessert', u'attributes-Good For-lunch', u'attributes-Good For-dinner',
u'attributes-Good For-breakfast', u'attributes-Good For-latenight', u'open', u'categories']

    # content = []
    # for row in result:
    # content.append([x[1] for x in row])
    # print content
    # content = [[]] for x in result]
    # print content
    # result = [content]
    # content = [str(x[1]) for x in result]
    # print len(result)
    FileHandler.WriteCSVContent(outfilename, headers_new, result)


def getStars(infilename, outfilename):
    headers = ["business_id", "stars"]
    result = []
    with open(infilename, "r") as f:
        for line in f:
            obj = json.loads(line)
            result.append([obj["business_id"], obj["stars"]])
    # print result
    FileHandler.WriteCSVContent(outfilename, headers, result)


def getReviews(infilename, outfilename):
    headers = ["business_id", "text"]
    result = []
    ids = getIds()
    with open(infilename, "r") as f:
        for line in f:
            obj = json.loads(line)
            if obj["business_id"] in ids:
                print obj["business_id"]
                result.append(
                    [obj["business_id"], obj["text"].replace("\n", "")])
    # print result
    FileHandler.WriteCSVContent(outfilename, headers, result)


def getIds():
    ids = []
    with open('ids.txt', 'r') as fr:
        for line in fr:
            business_id = line.strip('\n')
            ids.append(business_id)
    return ids


def ConvertLines(infilename, outfilename):
    result = []
    ids = getIds()
    for business_id in ids:
        content = []
        content.append(business_id.replace("\n", ""))
        with open(infilename, 'r') as fr:
            for line in fr:
                strs = line.split(',', 1)
                if len(strs) == 2:
                    if business_id == strs[0]:
                        content.append(strs[1].replace("\n", ""))
        result.append(copy.deepcopy(content))
    print len(result)
    FileHandler.WriteCSVContent(outfilename, [], result)


def FinailMerage(infilename_part1, infilename_part2, outfilename):
    with open(outfilename, 'w') as fw:
        with open(infilename_part2, 'r') as fr_2:
            with open(infilename_part1, 'r') as fr_1:
                fr_1.readline()
                header = fr_2.readline()
                fw.write(header)
                counter = 0
                while True:
                    line1 = fr_1.readline()
                    if line1:
                        line2 = fr_2.readline()
                        line_merage = ""
                        line_merage = line2.replace('\n', '')
                        strs1 = line1.split(',', 1)
                        if len(strs1) == 2:
                            line_merage += "," + strs1[1]
                            print line_merage
                        fw.write(line_merage.replace('\n', ''))
                        counter += 1
            else:
                break
        print counter

def GetLineWordsCount(infilename, outfilename):

    keywords = ["food", "good", "not", "so", "great", "very", "like", "service", "back", "really", "best", "only",
"nice", "well", "delicious", "pretty", "love", "better", "amazing", "fresh", "chicken", "menu", "definitely",
"sauce", "burger", "always", "cheese", "dinner", "everything", "never", "bar", "pizza", "experience", "steak",
"salad", "buffet", "fries", "friendly", "staff", "sushi", "lunch", "price", "worth", "server", "sure", "bad", "side",
"most", "taste", "drinks", "meat", "small", "hot", "beef", "both", "flavor", "fried", "awesome", "favorite",
"bread", "excellent", "dish", "big", "quality", "recommend", "new", "rice", "stars", "dessert", "sweet", "tasty",
"happy", "shrimp", "drink", "dishes", "special", "perfect", "soup", "ok", "sandwich", "hotel", "huge", "hour",
"overall", "prices", "super", "waiter", "friends", "location", "spicy", "pork", "served", "crab", "loved",
"atmosphere", "fish", "friend", "waitress", "chocolate", "maybe", "cream", "enjoyed", "wine", "decent",
"beer",
            "free", "lobster", "bacon", "star", "disappointed", "husband", "water", "high", "review", "busy",
"enjoy", "clean", "party", "coffee", "fun", "reviews", "burgers", "fast", "seafood", "perfectly", "thai", "tacos",
"quick", "fantastic", "liked", "wife", "french", "yummy", "absolutely", "crispy", "rib", "wrong", "expensive",
"wonderful", "appetizer", "pasta", "bbq", "salmon", "tea", "deal", "size", "start", "waited", "cheap", "cake",
"grilled", "fine", "butter", "mexican", "tuna", "desserts", "noodles", "flavorful", "music", "italian", "slow",
"seating", "chinese", "appetizers", "impressed", "expected", "hungry", "ambiance", "buffets", "pricey",
"sandwiches", "prime", "curry", "boyfriend", "quickly", "beautiful", "wow", "juicy", "reasonable", "asian",
"priced", "prepared", "share", "pieces", "horrible", "recommended", "vodka", "dirty", "comfortable",
"crowded", "cute", "incredible", "perfection", "disappointing", "interesting", "outstanding", "japanese",
"overpriced", "terrible", "yum"]
    result = []
    ids = getIds()
    progress = 0
    for my_id in ids:
        status = 0
        linecount = 0
        dic = {}
        with open(infilename, 'r') as fr:
            for line in fr:
                strs = line.split(',', 1)
                business_id = strs[0]
                if my_id == business_id:
                    linecount += 1
                    review_content = strs[1].lower()
                    for word in keywords:
                        pattern = re.compile(word)
                        searchresult = pattern.findall(review_content)
                        wordcount = 0
                        if searchresult:
                            wordcount = len(searchresult)
                        if word in dic:
                            dic[word] += wordcount
                        else:
                            dic.setdefault(word, wordcount)
                if linecount > 0 and my_id != business_id:
                    break

                    # print word, wordcount
        title = [my_id]
        wordcountlist = []
        if dic:
            wordcountlist = [dic[word] for word in keywords]
        wordcountlist = title + wordcountlist
        result.append(wordcountlist)
        progress += 1
        print str(progress) + ":" + str(linecount)
        # break
    newheader = ["business_id"] + keywords
    FileHandler.WriteCSVContent(outfilename, newheader, result)

if __name__ == '__main__':
    # converttocsv("yelp_academic_dataset_business.json", "final.csv")
    # getStars("yelp_academic_dataset_review.json", "stars.csv")
    # print getIds()
    # getReviews("yelp_academic_dataset_review.json", "review_mini.csv")
    # getReviews("mini.json", "review_mini.csv")
    # getMiniInfo("project_edited.csv", "out.csv")
    # ConvertLines("review_mini.csv", "review_mini_merage.csv")
    # FinailMerage('review_mini_merage.csv',
    #             'project_edited.csv', 'final_merage.csv')
    GetLineWordsCount("review_mini.csv", "wordcount.csv")
```

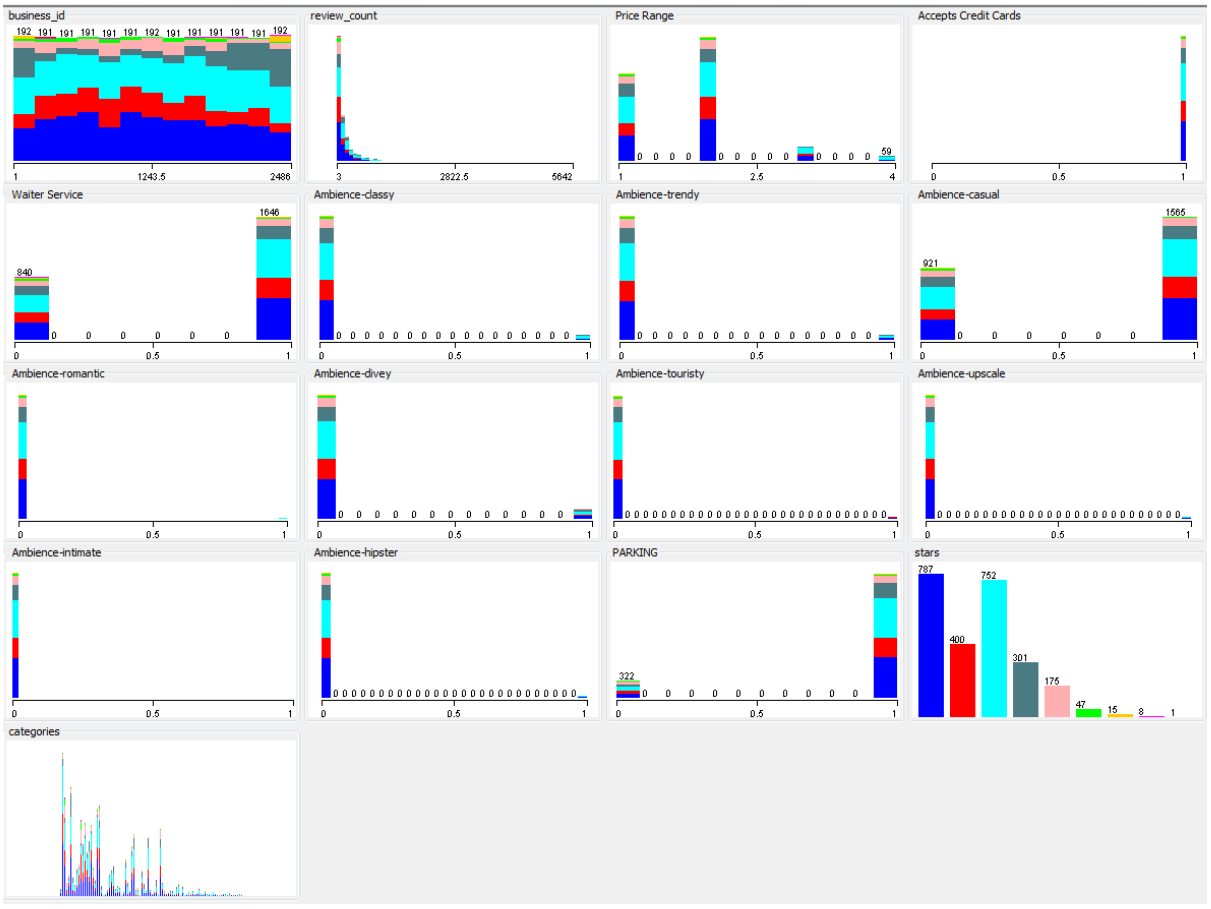
R

```
library(tm)
setwd(file.choose())
reviews <- read.csv(file.choose(), stringAsFactors=FALSE)
review_text <- past(reviews$text, collapse=" ")
review_source <- VectorSource(review_text)
corpus <- Corpus(review_source)
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, stripwhitespace)
corpus <- tm_map(corpus, removewords, stopwords(" English "))
content <- DocumentTermMatrix(corpus)
content1 <- as.matrix(dm)
frequency <- colSums(content1)
frequency <- sort(frequency, decreasing=TRUE)
```

Appendix 2 Category List

1.	Latin American	31.	Turkish	61.	Greek
2.	Nightlife Bars	32.	Szechuan, Chinese	62.	Breakfast & Brunch, Diners
3.	Pizza	33.	Asian Fusion	63.	Cuban
4.	Hotels	34.	Tapas Bars	64.	Event Planning
5.	BBQ	35.	Taiwanese, Chinese	65.	Creperies
6.	Mexican	36.	Sandwiches	66.	Tea
7.	Mediterranean	37.	Sushi Bars	67.	Ethiopian
8.	Filipino	38.	Cafes	68.	Hot Pot
9.	Delis	39.	Coffee & Tea	69.	Middle Eastern
10.	Pubs	40.	Poutineries	70.	Kosher
11.	Fast Food	41.	Korean	71.	Malaysian
12.	American (New)	42.	Vegetarian	72.	Ice Cream & Frozen Yogurt
13.	American (Traditional)	43.	Pakistani	73.	Afghan
14.	Seafood	44.	Steakhouses	74.	Bakeries, Filipino
15.	Thai	45.	Persian/Iranian	75.	Spanish
16.	Chinese	46.	Donuts	76.	Ramen, Japanese
17.	Buffets	47.	Diners	77.	Peruvian
18.	Cafes, restaurant	48.	Bakeries	78.	Bakeries, German
19.	Italian	49.	Grocery	79.	Caribbean
20.	Burgers	50.	Breakfast & Brunch	80.	British
21.	Hawaiian	51.	Wine Bars	81.	Brazilian
22.	Drugstores	52.	Casinos	82.	Do-It-Yourself Food
23.	Cantonese	53.	Meat	83.	Food Stands
24.	Hot Dogs	54.	Dim Sum	84.	Food Court
25.	Desserts	55.	Bagels	85.	Candy
26.	Vietnamese	56.	French	86.	Russian
27.	Japanese	57.	Cheesesteaks	87.	Bakeries, Italian
28.	Cajun/Creole	58.	Indian	88.	Moroccan
29.	Southern	59.	Juice Bars & Smoothies	89.	Brasseries
30.	Chicken	60.	Foundue	90.	Gluten-Free

Appendix 3 Attribute Visualisation



Appendix 4 Results from experiment

Review Dataset with 2 Classes Result

KNN (IBK)

=== Run information ===

```
Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -
A \"weka.core.EuclideanDistance -R first-last\""
Relation:     20 Attributes Normalised_Good Not so Good
Instances:    2486
Attributes:   21
              great
              best
              delicious
              amazing
              fresh
              definitely
              friendly
              flavor
              awesome
              favorite
              excellent
              fantastic
              yummy
              wonderful
              flavorful
              incredible
              yum
              recommend
              love
              perfect
              stars
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1819               73.1698 %
Incorrectly Classified Instances       667               26.8302 %
Kappa statistic                          0.4472
Mean absolute error                      0.27
Root mean squared error                  0.5156
Relative absolute error                 55.0939 %
Root relative squared error            104.1586 %
Total Number of Instances             2486
```

```
=== Detailed Accuracy By Class ===

              TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                0.794     0.351      0.75      0.794     0.771      0.721    Not So Good
                0.649     0.206      0.704     0.649     0.675      0.721    Good
Weighted Avg.   0.732     0.289      0.73      0.732     0.73       0.721
=== Confusion Matrix ===

    a    b   <-- classified as
 1126  292 |    a = Not So Good
  375  693 |    b = Good
```

Multinomial Naïve Bayes

```
=== Classifier model (full training set) ===
The independent probability of a class
--------------------------------------
Not So Good      0.5703376205787781
Good    0.4296623794212219

The probability of a word given the class
-----------------------------------------
        Not So Good     Good
great   0.17995404952761274  0.16741417626840832
best    0.07688770908507363  0.07177419692514325
delicious       0.05919867931188125  0.06916301777635304
amazing         0.03879763214518109  0.05225921321669513
fresh   0.06920221743008385  0.06455584860516679
definitely      0.055435790621351504        0.055471126039178434
friendly0.06318462642127298  0.055764427162108
flavor  0.06674683619461202  0.06285480307439381
awesome         0.03165026947445233  0.031816048145981185
favorite0.035307195229830786        0.03607043144698137
excellent       0.02824762847991321  0.029548894443252982
fantastic       0.01140383221306284  0.01343087199962977
yummy 0.01571955552650565  0.014968725645106788
wonderful       0.010576800207240424        0.013727501111111626
flavorful       0.011565536027637375        0.012203277696319521
incredible      0.0037984046611958654       0.0050023881921254345
yum     0.025611861267242866        0.02504330439050262
recommend       0.0497750443389862   0.04851010067350844
love    0.11681909755865551  0.11072448191660313
perfect 0.05011723427822517  0.05969716527144022

Time taken to build model: 0.03 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances       1418             57.0394 %
Incorrectly Classified Instances      1068             42.9606 %
Kappa statistic                   0
Mean absolute error                  0.487
```

```
Root mean squared error              0.492
Relative absolute error             99.3684 %
Root relative squared error          99.3892 %
Total Number of Instances           2486

=== Detailed Accuracy By Class ===

           TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
             1         1         0.57      1        0.726      0.665    Not So Good
             0         0         0         0        0          0.665    Good
Weighted Avg.   0.57      0.57      0.325     0.57      0.414      0.665

=== Confusion Matrix ===

    a    b   <-- classified as
 1418    0 |    a = Not So Good
 1068    0 |    b = Good
```

JRIP (RIPPER)

```
=== Classifier model (full training set) ===
JRIP rules:
===========
(delicious >= 0.160714) and (amazing >= 0.104693) => stars=Good (673.0/136.0)
(amazing >= 0.074074) and (friendly >= 0.133588) and (fresh >= 0.173913) => stars=Good
(129.0/44.0)
(great >= 0.359375) and (best >= 0.210526) and (definitely >= 0.15528) => stars=Good (62.0/15.0)
(love >= 0.230769) and (delicious >= 0.145038) and (definitely >= 0.147287) => stars=Good
(98.0/31.0)
(great >= 0.45283) and (favorite >= 0.118644) and (flavorful >= 0.015873) => stars=Good (28.0/4.0)
 => stars=Not So Good (1496.0/308.0)
Number of Rules : 6
Time taken to build model: 0.23 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances       1867             75.1006 %
Incorrectly Classified Instances      619             24.8994 %
Kappa statistic                    0.4865
Mean absolute error                  0.3478
Root mean squared error              0.4315
Relative absolute error             70.9632 %
Root relative squared error          87.1664 %
Total Number of Instances           2486
=== Detailed Accuracy By Class ===
           TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
             0.814     0.332     0.765     0.814     0.789      0.757    Not So Good
             0.668     0.186     0.73      0.668     0.697      0.757    Good
Weighted Avg.   0.751     0.27      0.75      0.751     0.749      0.757

=== Confusion Matrix ===
    a    b   <-- classified as
 1154  264 |    a = Not So Good
 355  713 |    b = Good
```

Linear Regression

```
=== Classifier model (full training set) ===
Linear Regression Model

stars =

     0.1871 * best +
     1.1401 * delicious +
     1.3187 * amazing +
     0.2186 * fresh +
     0.3153 * definitely +
     0.492  * friendly +
    -0.1945 * flavor +
     0.651  * awesome +
     0.7322 * favorite +
    -0.3479 * excellent +
     0.8227 * wonderful +
    -0.1034

Time taken to build model: 0.03 seconds

=== Cross-validation ===
=== Summary ===
Correlation coefficient                  0.5188
Mean absolute error                      0.3645
Root mean squared error                  0.4233
Relative absolute error                 74.3464 %
Root relative squared error             85.4751 %
Total Number of Instances             2486
```

Review Dataset with 9 Classes Result

KNN

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1046               42.0756 %
Incorrectly Classified Instances      1440               57.9244 %
Kappa statistic                          0.2347
Mean absolute error                      0.1295
Root mean squared error                  0.357
Relative absolute error                 76.4175 %
Root relative squared error            122.6567 %
Total Number of Instances             2486

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.488     0.281     0.445     0.488     0.466      0.603    class3.5
                 0.393     0.128     0.37      0.393     0.381      0.632    class3
```

```
            0.477     0.219      0.486     0.477     0.482      0.621    class4
            0.286     0.062      0.387     0.286     0.329      0.61     class4.5
            0.234     0.054      0.247     0.234     0.24       0.595    class2.5
            0.34      0.016      0.291     0.34      0.314      0.639    class2
            0.067     0.003      0.125     0.067     0.087      0.593    class5
            0.25      0.003      0.2       0.25      0.222      0.654    class1.5
            0         0          0         0         0          0.705    class1
Weighted Avg.    0.421     0.188      0.419     0.421     0.418      0.614

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   <-- classified as
 384 132 198  29  34   8   2   0   0 |   a = class3.5
 126 157  35   5  65  11   0   1   0 |   b = class3
 250  44 359  87   7   2   3   0   0 |   c = class4
  64   9 134  86   5   1   2   0   0 |   d = class4.5
  29  71   8   5  41  16   0   5   0 |   e = class2.5
   4  10   1   4  11  16   0   1   0 |   f = class2
   5   0   3   5   1   0   1   0   0 |   g = class5
   0   1   1   1   2   1   0   2   0 |   h = class1.5
   0   0   0   0   0   0   0   1   0 |   i = class1
```

Multinomial Naïve Bayes

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         850               34.1915 %
Incorrectly Classified Instances      1636               65.8085 %
Kappa statistic                          0.0386
Mean absolute error                      0.1688
Root mean squared error                  0.2904
Relative absolute error                 99.5784 %
Root relative squared error             99.7892 %
Total Number of Instances             2486

=== Detailed Accuracy By Class ===

            TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
            0.948     0.911      0.325     0.948     0.484      0.571    class3.5
            0         0          0         0         0          0.616    class3
            0.138     0.051      0.539     0.138     0.22       0.616    class4
            0         0          0         0         0          0.72     class4.5
            0         0          0         0         0          0.691    class2.5
            0         0          0         0         0          0.833    class2
            0         0          0         0         0          0.491    class5
            0         0          0         0         0          0.809    class1.5
            0         0          0         0         0          0.176    class1
Weighted Avg.    0.342     0.304      0.266     0.342     0.22       0.623

=== Confusion Matrix ===
  a   b   c   d   e   f   g   h   i   <-- classified as
 746   0  41   0   0   0   0   0   0 |   a = class3.5
 392   0   8   0   0   0   0   0   0 |   b = class3
 648   0 104   0   0   0   0   0   0 |   c = class4
 263   0  38   0   0   0   0   0   0 |   d = class4.5
 175   0   0   0   0   0   0   0   0 |   e = class2.5
  47   0   0   0   0   0   0   0   0 |   f = class2
  13   0   2   0   0   0   0   0   0 |   g = class5
   8   0   0   0   0   0   0   0   0 |   h = class1.5
   1   0   0   0   0   0   0   0   0 |   i = class1
```

JRIP (RIPPER)

```
JRIP rules:
===========
(amazing >= 0.5) => stars=class5 (3.0/1.0)
(great <= 0.230769) and (definitely <= 0.069892) and (love <= 0.181818) and (friendly >= 0.035714)
and (favorite <= 0.026882) and (friendly <= 0.09322) and (awesome <= 0.043011) and (delicious <=
0.117647) => stars=class2 (17.0/3.0)
(great <= 0.190164) and (great >= 0.117647) and (best <= 0.1) and (friendly <= 0.033333) =>
stars=class2 (11.0/4.0)
(delicious <= 0.08) and (great <= 0.29) and (definitely >= 0.019048) and (perfect <= 0.047619) and
(excellent >= 0.02139) and (delicious >= 0.03125) => stars=class2.5 (37.0/11.0)
(delicious <= 0.079545) and (great <= 0.287879) and (friendly >= 0.02381) and (friendly <= 0.14) and
(great <= 0.183673) and (perfect <= 0.014286) => stars=class2.5 (35.0/12.0)
(delicious >= 0.188119) and (friendly >= 0.16) and (amazing >= 0.136364) and (fantastic >= 0.017173)
and (perfect <= 0.27) => stars=class4.5 (101.0/31.0)
(delicious >= 0.180147) and (definitely >= 0.167116) and (best >= 0.214953) and (friendly >=
0.126716) and (perfect <= 0.25) => stars=class4.5 (34.0/11.0)
(amazing >= 0.113636) and (delicious >= 0.233696) and (perfect <= 0.2) and (awesome >= 0.102628)
=> stars=class4.5 (20.0/4.0)
(delicious <= 0.135) and (love <= 0.20155) and (perfect >= 0.023585) and (best <= 0.138518) =>
stars=class3 (195.0/97.0)
(amazing <= 0.068719) and (great >= 0.205882) and (favorite <= 0.058824) and (delicious <=
0.090753) and (delicious >= 0.05042) => stars=class3 (46.0/16.0)
(amazing >= 0.098485) and (love >= 0.317829) and (great >= 0.456667) and (yummy <= 0.058568) =>
stars=class4 (214.0/66.0)
(delicious >= 0.160714) and (great >= 0.410448) and (awesome >= 0.085487) => stars=class4
(150.0/58.0)
(definitely >= 0.147368) and (love >= 0.261905) and (best >= 0.166667) and (awesome >= 0.055556)
and (amazing <= 0.132622) => stars=class4 (52.0/12.0)
 => stars=class3.5 (1571.0/924.0)

Number of Rules : 14
Time taken to build model: 1.13 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        992               39.9035 %
Incorrectly Classified Instances     1494               60.0965 %
Kappa statistic                          0.1635
```

```
Mean absolute error                 0.155
Root mean squared error               0.2869
Relative absolute error             91.4254 %
Root relative squared error           98.5678 %
Total Number of Instances            2486

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.705     0.543     0.376     0.705     0.49      0.598    class3.5
                 0.155     0.058     0.341     0.155     0.213     0.682    class3
                 0.374     0.181     0.472     0.374     0.417     0.639    class4
                 0.236     0.038     0.461     0.236     0.312     0.75     class4.5
                 0.12      0.016     0.362     0.12      0.18      0.717    class2.5
                 0.043     0.006     0.118     0.043     0.063     0.722    class2
                 0         0.001     0         0         0         0.615    class5
                 0         0         0         0         0         0.705    class1.5
                 0         0         0         0         0         0.288    class1
Weighted Avg.    0.399     0.242     0.4       0.399     0.367     0.653

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   <-- classified as
 555  51 151  17  10   2   1   0   0 |   a = class3.5
 296  62  21   1  15   5   0   0   0 |   b = class3
 390  18 281  59   2   1   1   0   0 |   c = class4
  84   8 137  71   0   0   1   0   0 |   d = class4.5
 113  30   3   1  21   7   0   0   0 |   e = class2.5
  29   8   0   0   8   2   0   0   0 |   f = class2
   7   1   2   5   0   0   0   0   0 |   g = class5
   2   4   0   0   2   0   0   0   0 |   h = class1.5
   1   0   0   0   0   0   0   0   0 |   i = class1
```

Linear Regression

```
stars =

      0.1967 * great +
      0.3454 * best +
      1.6843 * delicious +
      1.4815 * amazing +
      0.3295 * fresh +
      0.4197 * definitely +
      0.6034 * friendly +
     -0.3844 * flavor +
      1.0318 * awesome +
      0.8935 * favorite +
     -0.622  * fantastic +
      0.7689 * flavorful +
     -1.2457 * incredible +
      0.3017 * love +
      2.7446

Time taken to build model: 0.18 seconds
=== Cross-validation ===
=== Summary ===

Correlation coefficient                 0.6211
Mean absolute error                      0.3644
Root mean squared error                  0.4767
Relative absolute error                 75.6691 %
Root relative squared error             78.3397 %
Total Number of Instances              2486
```

Business Dataset with 2 Classes Result

OneR

```
review_count:
	< 12.5	-> Not So Good
	< 13.5	-> Good
	< 22.5	-> Not So Good
	< 23.5	-> Good
	< 36.5	-> Not So Good
	< 37.5	-> Good
	< 44.5	-> Not So Good
	< 45.5	-> Good
	< 52.5	-> Not So Good
	< 53.5	-> Good
	< 61.5	-> Not So Good
	< 62.5	-> Good
	< 73.5	-> Not So Good
	< 74.5	-> Good
	< 78.5	-> Not So Good
	< 79.5	-> Good
	< 80.5	-> Not So Good
	< 81.5	-> Good
	< 94.5	-> Not So Good
	< 96.5	-> Good
	< 108.5 -> Not So Good
	< 113.5 -> Good
	< 120.5 -> Not So Good
	< 123.5 -> Good
	< 125.5 -> Not So Good
	< 128.5 -> Good
	< 132.5 -> Not So Good
	< 135.5 -> Good
	< 147.5 -> Not So Good
	< 149.5 -> Good
	< 152.5 -> Not So Good
	< 159.5 -> Good
	< 163.5 -> Not So Good
	< 169.5 -> Good
	< 172.5 -> Not So Good
```

```
        < 174.5 -> Good
        < 184.5 -> Not So Good
        < 186.5 -> Good
        < 189.5 -> Not So Good
        < 201.5 -> Good
        < 204.5 -> Not So Good
        < 211.5 -> Good
        < 222.5 -> Not So Good
        < 226.5 -> Good
        < 236.5 -> Not So Good
        < 242.5 -> Good
        < 245.5 -> Not So Good
        < 249.5 -> Good
        < 255.5 -> Not So Good
        < 264.5 -> Good
        < 269.5 -> Not So Good
        < 284.5 -> Good
        < 294.5 -> Not So Good
        < 301.5 -> Good
        < 343.5 -> Not So Good
        < 348.5 -> Good
        < 358.5 -> Not So Good
        < 368.5 -> Good
        < 374.5 -> Not So Good
        < 382.5 -> Good
        < 392.5 -> Not So Good
        < 407.0 -> Good
        < 426.5 -> Not So Good
        < 443.5 -> Good
        < 454.5 -> Not So Good
        < 488.5 -> Good
        < 502.0 -> Not So Good
        < 545.5 -> Good
        < 562.5 -> Not So Good
        < 979.5 -> Good
        < 1052.0        -> Not So Good
        < 1362.5        -> Good
        < 1417.5        -> Not So Good
        >= 1417.5       -> Good
(1643/2486 instances correct)


Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1445               58.1255 %
Incorrectly Classified Instances      1041               41.8745 %
Kappa statistic                          0.1097
Mean absolute error                      0.4187
```

```
Root mean squared error              0.6471
Relative absolute error             85.441  %
Root relative squared error          130.723  %
Total Number of Instances           2486

=== Detailed Accuracy By Class ===

            TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
              0.756     0.651      0.607     0.756     0.673      0.553    Not So Good
              0.349     0.244      0.519     0.349     0.417      0.553    Good
Weighted Avg.    0.581     0.476      0.569     0.581     0.563      0.553

=== Confusion Matrix ===

    a    b   <-- classified as
 1072  346 |    a = Not So Good
  695  373 |    b = Good
```

Naïve Bayes

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       1548              62.2687 %
Incorrectly Classified Instances      938              37.7313 %
Kappa statistic                     0.1851
Mean absolute error                  0.418
Root mean squared error              0.4906
Relative absolute error             85.2882 %
Root relative squared error          99.1093 %
Total Number of Instances           2486

=== Detailed Accuracy By Class ===

            TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
              0.838     0.663      0.627     0.838     0.717      0.666    Not So Good
              0.337     0.162      0.61      0.337     0.434      0.666    Good
Weighted Avg.    0.623     0.448      0.62      0.623     0.596      0.666

=== Confusion Matrix ===

    a    b   <-- classified as
 1188  230 |    a = Not So Good
  708  360 |    b = Good
```

J48

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       1557              62.6307 %
```

```
Incorrectly Classified Instances       929              37.3693 %
Kappa statistic                          0.223
Mean absolute error                      0.4257
Root mean squared error                  0.5067
Relative absolute error                 86.8515 %
Root relative squared error            102.3533 %
Total Number of Instances             2486

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.729     0.51       0.655     0.729     0.69       0.642    Not So Good
                 0.49      0.271      0.577     0.49      0.53       0.642    Good
Weighted Avg.    0.626     0.407      0.621     0.626     0.621      0.642

=== Confusion Matrix ===

    a    b   <-- classified as
 1034  384 |    a = Not So Good
  545  523 |    b = Good
```

Linear Regression

```
=== Classifier model (full training set) ===

Linear Regression Model

stars =

      0.0002 * review_count +
     -0.0333 *  Price Range +
     -0.0487 *  Waiter Service +
      0.2583 *  Ambience-classy +
      0.1989 *  Ambience-romantic +
      0.1606 *  Ambience-divey +
     -0.3616 *  Ambience-touristy +
      0.1747 *  Ambience-intimate +
      0.0716 * PARKING +
      0.3917

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.1954
Mean absolute error                      0.4689
Root mean squared error                  0.4859
Relative absolute error                 95.651  %
Root relative squared error             98.1279 %
Total Number of Instances             2486
```

Business Dataset with 9 Classes Result

OneR

Test mode:10-fold cross-validation

```
=== Classifier model (full training set) ===

review_count:
	< 4.5	-> class3.5
	< 7.5	-> class3
	< 10.5	-> class3.5
	< 11.5	-> class3
	< 12.5	-> class3.5
	< 13.5	-> class4
	< 14.5	-> class3
	< 15.5	-> class3.5
	< 16.5	-> class3
	< 21.5	-> class3.5
	< 25.5	-> class4
	< 28.5	-> class3.5
	< 29.5	-> class3
	< 35.5	-> class3.5
	< 36.5	-> class3
	< 37.5	-> class4
	< 40.5	-> class3.5
	< 41.5	-> class4
	< 44.5	-> class3.5
	< 45.5	-> class4
	< 47.5	-> class3.5
	< 49.5	-> class3
	< 51.5	-> class3.5
	< 52.5	-> class3
	< 53.5	-> class4
	< 56.5	-> class3.5
	< 57.5	-> class3
	< 61.5	-> class3.5
	< 64.5	-> class4
	< 67.5	-> class3.5
	< 70.5	-> class3
	< 78.5	-> class3.5
	< 79.5	-> class4
	< 103.5 -> class3.5
	< 106.5 -> class4
	< 109.5 -> class3.5
	< 113.5 -> class4
	< 120.5 -> class3.5
	< 124.5 -> class4
	< 125.5 -> class3.5
	< 133.5 -> class4
	< 137.5 -> class3
	< 144.5 -> class3.5
```

```
        < 147.5 -> class4
        < 152.5 -> class3.5
        < 158.5 -> class4
        < 163.5 -> class3.5
        < 167.5 -> class4
        < 173.5 -> class3.5
        < 185.5 -> class4
        < 189.5 -> class3.5
        < 206.5 -> class4
        < 225.5 -> class3.5
        < 229.5 -> class4
        < 236.5 -> class3.5
        < 242.5 -> class4
        < 247.5 -> class3.5
        < 252.5 -> class4
        < 255.5 -> class3.5
        < 283.5 -> class4
        < 293.5 -> class3.5
        < 308.0 -> class4
        < 315.5 -> class3.5
        < 325.5 -> class4
        < 343.5 -> class3.5
        < 359.5 -> class4
        < 374.5 -> class3.5
        < 382.5 -> class4
        < 392.5 -> class3.5
        < 482.5 -> class4
        < 502.0 -> class3.5
        < 679.0 -> class4
        < 715.0 -> class4.5
        < 1329.0        -> class4
        < 1917.0        -> class3.5
        >= 1917.0       -> class4
(1019/2486 instances correct)

Time taken to build model: 0.04 seconds
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        775             31.1746 %
Incorrectly Classified Instances     1711              68.8254 %
Kappa statistic                    0.0212
Mean absolute error                  0.1529
Root mean squared error                0.3911
Relative absolute error              90.2179 %
Root relative squared error          134.3706 %
Total Number of Instances            2486

=== Detailed Accuracy By Class ===

        TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
```

```
           0.554    0.535     0.324    0.554    0.409     0.509    class3.5
           0.09      0.073     0.19      0.09      0.122      0.508    class3
           0.402    0.356     0.328    0.402    0.361     0.523    class4
           0.003    0.011     0.04     0.003    0.006     0.496    class4.5
           0         0.003     0         0         0          0.498    class2.5
           0         0          0         0         0          0.5      class2
           0         0          0         0         0          0.5      class5
           0         0          0         0         0          0.5      class1.5
           0         0          0         0         0          0.5      class1
Weighted Avg.   0.312    0.291     0.237    0.312    0.259     0.511

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   <-- classified as
 436  60 281   7   3   0   0   0   0 |   a = class3.5
 235  36 126   3   0   0   0   0   0 |   b = class3
 397  41 302   9   3   0   0   0   0 |   c = class4
 140  29 130   1   1   0   0   0   0 |   d = class4.5
  94  17  61   3   0   0   0   0   0 |   e = class2.5
  30   4  11   2   0   0   0   0   0 |   f = class2
  10   0   5   0   0   0   0   0   0 |   g = class5
   3   2   3   0   0   0   0   0   0 |   h = class1.5
   0   0   1   0   0   0   0   0   0 |   i = class1
```

J48

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         807               32.4618 %
Incorrectly Classified Instances      1679               67.5382 %
Kappa statistic                          0.0831
Mean absolute error                      0.1588
Root mean squared error                  0.3205
Relative absolute error                 93.6684 %
Root relative squared error            110.1118 %
Total Number of Instances             2486

=== Detailed Accuracy By Class ===

         TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
           0.461    0.411     0.342    0.461    0.393     0.536    class3.5
           0.203    0.123     0.24     0.203    0.22      0.594    class3
           0.384    0.279     0.374    0.384    0.379     0.579    class4
           0.146    0.064     0.239    0.146    0.181     0.622    class4.5
           0.171    0.039     0.252    0.171    0.204     0.631    class2.5
           0         0.005     0         0         0          0.628    class2
           0         0          0         0         0          0.471    class5
           0         0          0         0         0          0.485    class1.5
           0         0          0         0         0          0.496    class1
Weighted Avg.   0.325    0.245     0.307    0.325    0.311     0.576
```

```
=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   <-- classified as
 363 106 242  53  19   4   0   0   0 |   a = class3.5
 177  81  90  13  36   3   0   0   0 |   b = class3
 312  67 289  68  16   0   0   0   0 |   c = class4
 112  24 118  44   3   0   0   0   0 |   d = class4.5
  69  45  25   2  30   4   0   0   0 |   e = class2.5
  20  10   5   1  11   0   0   0   0 |   f = class2
   6   2   4   2   1   0   0   0   0 |   g = class5
   2   2   0   1   2   1   0   0   0 |   h = class1.5
   0   0   0   0   1   0   0   0   0 |   i = class1
```

Naive Bayes

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         816               32.8238 %
Incorrectly Classified Instances      1670               67.1762 %
Kappa statistic                          0.0937
Mean absolute error                      0.1638
Root mean squared error                  0.2987
Relative absolute error                 96.5922 %
Root relative squared error            102.6247 %
Total Number of Instances             2486

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.69      0.57       0.359     0.69      0.473      0.58     class3.5
                 0.055     0.044      0.193     0.055     0.086      0.573    class3
                 0.267     0.129      0.474     0.267     0.342      0.616    class4
                 0.113     0.033      0.321     0.113     0.167      0.657    class4.5
                 0.051     0.013      0.237     0.051     0.085      0.663    class2.5
                 0.128     0.045      0.052     0.128     0.074      0.738    class2
                 0         0.02       0         0         0          0.78     class5
                 0.125     0.05       0.008     0.125     0.015      0.873    class1.5
                 0         0.002      0         0         0          0.118    class1
Weighted Avg.    0.328     0.232      0.345     0.328     0.294      0.61

=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   i   <-- classified as
 543  36 108  27  10  32  10  20   1 |   a = class3.5
 262  22  40   8  12  24   9  22   1 |   b = class3
 435  22 201  30   3  18  15  26   2 |   c = class4
 152  10  60  34   1  12  11  21   0 |   d = class4.5
  91  18  13   4   9  19   2  19   0 |   e = class2.5
  18   5   1   0   3   6   2  12   0 |   f = class2
   7   0   1   2   0   2   0   3   0 |   g = class5
   2  1  0  1  0  3  0  1  0 |   h = class1.5
   1  0  0  0  0  0  0  0  0 |   i = class1
```

Linear Regression

```
stars =

      0.0002 * review_count +
     -0.0675 *  Price Range +
      0.3529 *  Ambience-classy +
      0.1876 *  Ambience-trendy +
      0.0687 *  Ambience-casual +
      0.2555 *  Ambience-romantic +
      0.2778 *  Ambience-divey +
     -0.4244 *  Ambience-touristy +
      0.1417 *  Ambience-upscale +
      0.2732 *  Ambience-intimate +
      0.1414 *  Ambience-hipster +
      0.1675 * PARKING +
      3.4297

Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.2276
Mean absolute error                       0.4659
Root mean squared error                     0.5924
Relative absolute error                 96.7513 %
Root relative squared error               97.3635 %
Total Number of Instances                 2486
```