CMPS 242 Project Fall 2016 : Predicting Yelp business ratings using Multiclass Classification

Greeshma Swaminathan GSWAMINA@UCSC.EDU
Neha Ojha NOJHA@UCSC.EDU
Jianshen Liu JLIU120@UCSC.EDU
https://github.com/ljishen/yelp-dataset-challenge
University of California, Santa Cruz, 1156 High Street, Santa Cruz, CA 95064

Project Statement

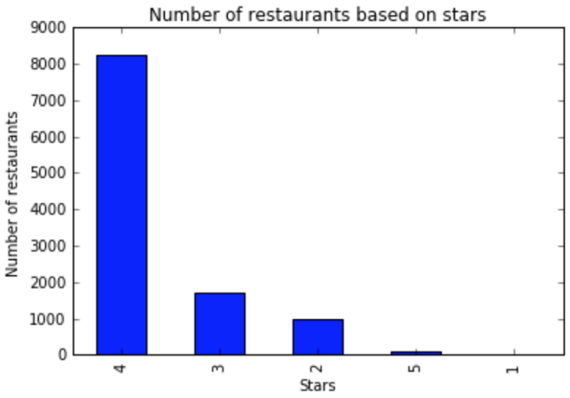
The aim of our project is to predict restaurant ratings in the Yelp dataset using multiclass classification. The degree of success of restaurants or the label that we are predicting is the attribute: "stars", which is one of the attributes provided with the business dataset. As a part of our initial analysis, we had only considered useful features from the business dataset(like city, attributes, open hours) in order to make predictions. In our final analysis, we have incorporated a number of features that were derived from other datsets like tip, review, checkin etc. We have found that using some of these features has greatly increased the accuracy of our classifier (the best we got is **94%**) while a few features aren't as useful. We have made comparisons and listed our observations in the Results section.

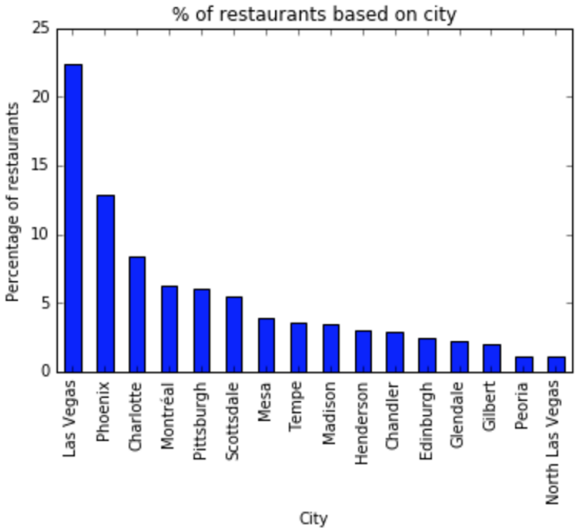
The problem that we are trying to solve is a multiclass classification problem, with stars as the labels. We have experimented with 10 floating point labels for stars(.5,1,1.5 etc till 5) and also 5 labels with rounded stars - (1,2,3,4,5). We concluded that rounded stars give us the best performance . We have implemented the **Multiclass Naive Bayes classifier**(1) for our multiclass classification problem. We compared our classifier performance with Scikit learn's Gaussian Naive Bayes classifier. Carefully selecting features gives us better performance than Scikit-learn's Gaussian Naive Bayes classifer. We have also done sentiment analysis on the reviews and tips using Stanford CoreNLP (2) and used features generated from it, as inputs to our classifier. A detailed explanation of the experimental setup and observations can be found in the Evaluation and Results section.

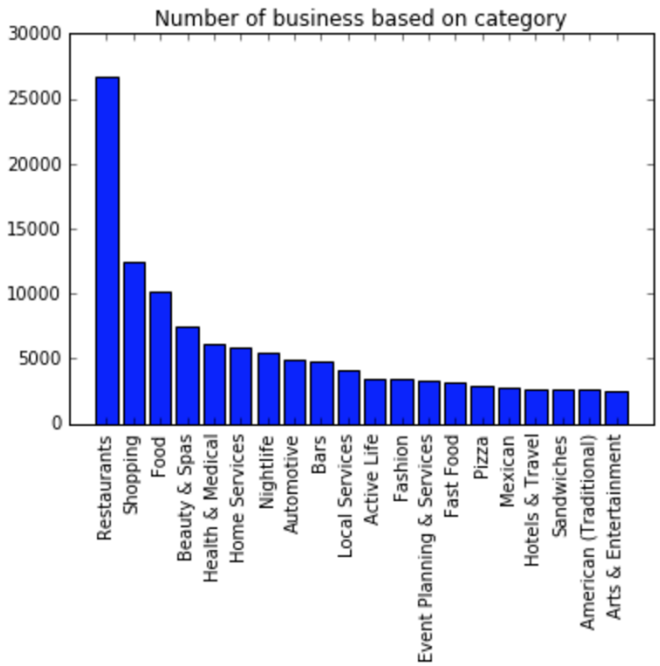
Feature Engineering

0.1. Preprocessing

We are using Python for the implementation. The JSON files were loaded into pandas dataframes. This helped us to get a quick overview of the data. Since 'Restaurants' form the majority of the data, we have focused our initial experiments on restaurants (around 11,042 entries - accuracy 94 %) by filtering the other businesses out. We have also done a run on all businesses (above 70,000 entries - accuracy of 90 %). We also filtered the data to get only the 'Open' businesses.







New attribute	Values
Credit Cards Accepted	True,False, UnKnown
Alcohol	Full_bar, beer_and_wine, None, Unknown
Take out	True,False, UnKnown
Noise level	Very_Loud, Loud, Average, Quiet, UnKnown
Price Range	1,2,3,4,Unknown
Caters	True,False, UnKnown

Table 1. Features generated from business 'attributes' column

Dataset	New attribute
Reviews	review_sentiment_rating combined with useful
Tip	tip_rating combined with number of likes
Checkin	checkin_count
Review	review_star_rating mean of review stars

Table 2. Features generated from other datasets

0.2. Visualization

We have done some basic visualizations to know the dataset better. The figure "Number of restaurants based on stars" shows clearly that most of the restaurants have a rating in between 3.5-4, while there are very less 1 star restaurants.

The figure "% of restaurants based on city" shows the percentage of restaurants in various cities.For the sake of cleanliness, we have not shown cities with less than 1% restaurants. Las Vegas has most number of restaurants.

The figure "Number of businesses based on category" shows that restaurants form the major chunk of the businesses in the given dataset.

0.3. Feature generation

Most of the datasets have nested relevant attributes, therefore we transformed data by decomposing in to new features. Similarly for the attribute 'hours' we have generated features like breakfast, lunch, night, weekend etc. For the user dataset, features like compliments and votes which have multiple sub entries we have broken them into separate features. Refer Table 1 for the details on nested features in the business data set. Features generated from other tables are mentioned in Table 2.

We have tried to incorporate most the data that has been provided with yelp_dataset_challenge_academic dataset in order to make our classifier more robust. We have created features using the review, tip and checkin datasets. These features have been used as attributes for the Naive Bayes classifier. Following are the complete list of features that we used. A discussion on the most relevant features is included in the results section.

1. City - from business data set
2. Review count - from business data set
3. Review sentiment rating – Generated feature using sentiment analysis on reviews from reviews data set
4. Review star rating - Stars given for individual reviews from the reviews data set.
5. Tip rating - Generated feature using sentiment analysis on reviews from tips data set
6. Number of reviews per checkin from the checkin and business data set
7. Hours from the business data set
8. Accepts Credit Cards - from the attributes column of business data set.
9. Alcohol - from the attributes column of business data set
10. Caters - from the attributes column of business data set
11. Noise Level - from the attributes column of business data set
12. Price Range - from the attributes column of business data set
13. Take-out - from the attributes column of business data set

0.4. Sentiment analysis

Feedback (e.g. reviews and tips) from customers are very important for the rating of a business. We noticed that businesses with negative feedback are usually having a low rating. Thus, we want to use the sentiment of feedback as a feature for our classifier. The `SentimentAnnotator` provided by Stanford CoreNLP can produce 5 different results which represented by different integers range from 0 to 4 for one sentence: VERY NEGATIVE, NEGATIVE, NEUTRAL, POSITIVE and VERY POSITIVE. There are two problems when we were using this tool for sentiment analysis. First, reviews or tips are usually comprised of more than one sentences, we need to merge all the sentiment values from all sentences to give a final score for this feedback . Second, even though this sentiment analyzer claims to have 80.7% accuracy, we found some unreasonable examples while we testing this tool. For instance, it gives POSITIVE result for sentence "I'm not happy after this lunch!", while NEGATIVE result for sentence "I'm sad after this lunch!". To solve the first problem, we first move the output range from [0, 4] to [-2, 2] to ensure the NEUTRAL value don't have impact when we calculate the average sentiment value for a review. We express our merge algorithm in the formula as follows, V denotes the value space of the output from sentiment analysis, s_v denotes the number of sentences have sentiment value of v, S denotes the number of sentences in a review,

$$\frac{\sum_V v \cdot s_v}{S}$$

Therefore, we got one sentiment score for each of review as well as tip. For the second problem, we can verify the accuracy of sentiment analysis by comparing our classifier performance with or without the sentiment feature.

0.5. Digitizing continuous features - sentiment values , average stars and review per checkin.

We use bins categorization to digitize features with continuous value to make it easier to integrate into our Naive Bayes classifier. These continuous features include sentiment score, average stars of reviews from a business and review per check-in of a business. We found that the number of bins for categorizing continuous value has significant impact on the classifier performance. Specifically, the more bins the higher performance, but when the number of bins is more than 100, the positive impact starts to reduce. This is because granularity is already fine enough to capture the difference of rating for business. There is one exception , for the feature of review per check-in of a business, no matter how many bins we use, it degrades the classification accuracy, so we finally dropped this feature.

Algorithm/Model Formulation

0.6. Theory

We are using Naive Bayes multiclass classifier for the classification. Naive Bayes classifier uses Bayes theorem for prediction between classes. 'Naive' indicates the assumption that the features are independent of each other.

It considers each attribute and class label as random variables. Given a record with attributes $A_1, A_2...A_n$, the goal is to predict class C, which for our case is 'stars'. This can be estimated directly from the data. As per Baye's rule, we can write the posterior probability of each class as below.
$P(C|A_1, A_2, , A_n) \propto P(A_1, A_2, , A_n|C).P(C)$
And with the Naive Baye's assumption this becomes $P(C|A_1, A_2, , A_n) \propto \prod_{i=1}^{N} P(A_i|C).P(C)$ We calculate the posterior probability estimate for each class and then pick the class that has the maximum probability.
$argmax_{k=1}^{K} P(C_k|A_1, A_2, , A_n)$ where k is the number of classes

We have divided the data into training and test sets by us-

ing train-test split as well as K-fold cross-validation(k=10). We use the training set to train the classifier and evaluate the performance of the classifier by comparing the results produced by the classifier with true labels of the test set.

0.7. Math

To calculate the feature vector we follow the following approach. First, a group by is done on the data with the attribute 'stars'. For each star group, we find the number of restaurants matching each dimension (frequency) and calculate the probability as frequency/total number of restaurants in this class. Additionally to include Laplace smoothing we add +1 to the numerator and number of unique dimension values to the denominator.

Evaluation

The performance of our classifier can be determined by measuring the number of correct predictions that it makes. For example, M is the number of correct prediction in terms of the attribute of star, N is the total number of restaurants in the test set, then the success ratio is $\frac{M}{N}$.

Initial Analysis: We separated the business data set into two parts, 80% as the training data and rest as the test data. By only using 8 attributes of the business data set and for the best run, we achieved **27.2% of accuracy**, which was better than the random result but was not good enough. However, we also calculated the **average of absolute distance** between our predictive result and the true star, and it gave **0.693**, which meant for those missed predictions they are actually very close to the true values. Therefore, it was reasonable to assume that our model had scope for vast improvement by investigating more of other features/attributes, e.g. user reviews and tips. We used the following techniques to improve the performance of our classifier and made a comparison using Scikit-learn.

Improvements:

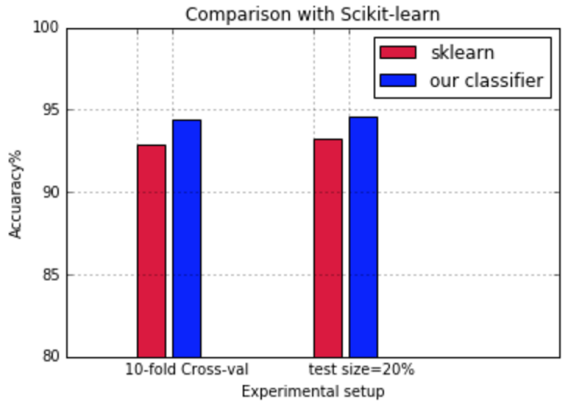
- Feature selection and parameter tuning

  A key component of getting better performance from our classifier has been selecting features that have a positive impact. We have created a bunch of features using the additional datasets, however, only some of the these features have been useful. To start with further improvements, we integrated review sentiment analysis both from user reviews and user tips. This gave our classifier a good boost in performance and put the accuracy roughly at around **45%** for 10 labels. Since we started using the reviews, we observed that several restaurants that are having very few reviews are skewing the performance of our classifier. To correct this we introduced a filter for the review count and selected only restaurants with more than **20** reviews. This improved our classifier performance to around **54 %**. At this point we observed that for the misclassified labels, the average distance from the correct label was around .16 which is a very small number. So to tune the classifier, we decided to go with broader bins for classification label and used rounded stars as classification label. This gave us an astonishing increase in accuracy to **84 %**. The final feature which gave another 10% improvement is the 'review star' rating , which is the individual star ratings given by user for each review. This finally gave the classifier an accuracy of **94 %**. To conclude, the most useful features for our classifier is review sentiment rating, tip sentiment rating and review stars.

- Using Cross-validation

  Train test split may sometimes have a lot of variation in performance for different sample of data. K-fold cross validation reduces this variance by averaging over k different partitions, so the performance estimate is less sensitive to the partitioning of the data. We have used 10-fold cross-validation in order to deal with this issue.

- Comparison with Scikit-learn modules

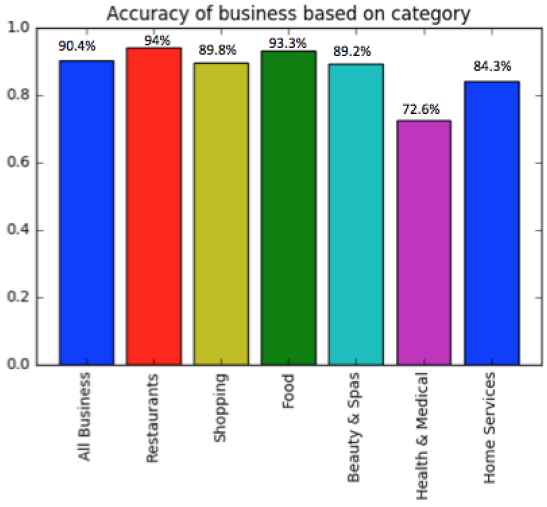


  We have provided a comparison of our classifier's performance with Scikit-learn . We created a specific experimental setup(Case4 in Table 3) and analyzed the performance of both our classifier and Scikit-learn's Gaussian Naive Bayes classifier in predicting the stars. We used the following attributes - checkin_rating, city, review_count, review_sentiment_rating, review_star_rating and tip_rating as the attributes to train a Gaussian Naive Bayes classifier and the label was the stars attribute. We encoded the city column using LabelEncoder from the preprocessing module in Scikit-learn . We have used rounded stars label. Using a train test split with test size = 0.2, we performed multiclass classification with Gaussian Naive Bayes classifier.

Case	Experimental setup	Accuracy
1	With rounded stars and review_sentiment_rating and test-size=0.2	94.34%
2	Without review_star_rating for rounded stars and test-size=0.2	78.98%
3	With rounded stars and review_sentiment_rating and 10-fold cross val	94.27%
4	With rounded stars and with working_type and attributes and 10-fold cross val	94.4%
5	Case 4 using Scikit-learn	92.87%
6	Case 4 using Scikit-learn and with test size=0.2	93.2%
7	Without rounded stars and test size=0.2	87.5%

Table 3. Summary of Results

This gave us an accuracy of 93.2%. Secondly, we performed a 10-fold cross-validation and generated the mean accuracy of the Gaussian Naive classifier. This resulted in an accuracy score of 92.8%. The figure "Comparison with Scikit-learn" shows the results. In both scenarios, our classifier performs better.



Results

Table 3 summarizes the most important results of our analysis based on our assumptions mentioned in the Experimental setup column. We have shown one experiment without rounded stars because overall rounded stars gives us better accuracy values.

The figure "Accuracy of business based on category", gives the overall accuracy of our classifier for a number of different business categories. The accuracy considering all the businesses is 90.4%. The accuracy of our classifier in predicting star values for only restaurants is the highest-94%.

One of the attributes that did not give us much improvement in accuracy was checkin_count, which was derived from the checkin dataset. This could probably because we created a feature Number of reviews per checkin, which may not add much value since reviews and checkins do not have direct relationship. For example, high rated business could have high ratio of reviews because customers want to show their good experience, while low rated business could also have high ratio of reviews because they want to complain the experience.

The Review star rating can be seen as a biased feature, because Yelp probably uses some algorithm using this parameter to calculate star ratings for businesses. Therefore, we have performed an experiment, Case 2 in Table 3, by removing this feature from our feature set. We can observe that eliminating it reduces the accuracy score. Furthermore, based on our experimental setup, it shows that with rounded stars gives a better accuracy of the classifier.

Future work

In our initial analysis, we had only used the attributes from the business dataset in order to get a baseline performance for our classifier. Later, we used various ways like 1. adding other attributes from reviews, tip, checkin datasets 2. scaling and normalizing the data 3. tuning parameters 4. using cross validation to improve the accuracy of our classifier. At this point, we have generated pretty good accuracy scores with respect to predicting 'stars' of restaurants. The key to success was carefully selecting features that are important for our multiclass classification problem. We also hope to see interesting results by incorporating useful attributes from the user dataset as features for the Naive Bayes classifier. We also could perform text mining from scratch on the 'reviews' and 'tips' of users, in order to categorize the restaurants as 'good' or 'bad'. Hopefully, this will be a fun experience.

References

Naive Bayes Classifier. `https://en.wikipedia.org/wiki/Naive_Bayes_classifier`.

Stanford CORE NLP. `http://stanfordnlp.github.io/CoreNLP/index.html`.