Sentiment Analysis of User Reviews
May 3rd, 2019
Nathan Cooper Jones, Eric Ballam

Abstract

In today's digital age, almost all of our decisions are backed by the support of some sort of user-based review system, most of which are solely in a free-form text format. In this paper, we compare the performance of three different classification models when attempting to classify the general rating (positive or negative) of Yelp reviews. Using the text of the review as the only model input, we assessed performance on Naive Bayes models using 1, 2, and 3 n-gram values, logistic regression with a sentiment analysis backing, and a rather experimental model stemming from the field of information retrieval: a TF-IDF ranked k-nearest neighbors model. After running the models with robust training and testing datasets, we found that Naive Bayes using 3-gram sequences outperformed the other models with an accuracy rate of 90.30%.

Introduction

In today's digital age, almost all of our decisions are backed by the support of some sort of user-based review system, whether it be the product we purchase, the restaurant we dine in, the movie we decide to view, the school we attend, and more. While certain sites such as Yelp and Amazon provide the use of a star-system and to summarize the overall user rating as well as tags other users can mark based on the helpfulness of the review, most reviews are of free-form text, with the only way to gauge sentiment by manually reading the review. As many of the methods and models in the field of statistical learning are based around numerical features, we attempt the challenge of analyzing this unstructured data to draw meaningful conclusions.

In this paper, we apply methods from this course towards the Yelp dataset, which contains both the review stars (with a score ranging from 1 to 5) and the review text, in order to develop a comprehensive model that can accurately predict the sentiment of a review based on the text only. We examine three different models to accomplish this: a simple Naive Bayes model based on n-grams of the text, a logistic regression model used on the sentiment of the review determined by a standard natural-language processing algorithm, and an experimental model combining the weighting system often used in information retrieval, the TF-IDF rank, and the k-nearest neighbor model. We detail these methods, their implementation, and results in the upcoming sections.

Data Sources

The data for this project was gathered from Yelp's 13th round of their Yelp Dataset Challenge, made available through Kaggle. This data includes information on businesses, users, reviews, tips, and "check-ins" in ten different cities across three separate countries. It was available from January 15, 2019 until December 31, 2019. For our purposes we only were concerned with the Review dataset. This dataset contains millions of unique reviews and has fields on review id, user id, business id, number of stars, date of the review, text and the number of times the review was found useful, funny, or cool.

Proposed Methodology

From the Yelp dataset, we used the star-rating and free-form text for training the models and only the latter for testing, which involved us dropping the “date” column as well as additional parameters measuring whether other users thought the review was “funny,” “cool,” or “useful,” as most reviews tend to only consist of text with no other attributed, and we attempted to model this situation as closely as possible when training a model for this purpose. In an effort to simplify model complexity, we created a new column, *adjStars*, which was a binary dummy variable representing whether the review was positive (4 or 5 stars) or negative (1, 2, or 3 stars), which was used for testing in our predictive models rather than the raw star count. The original dataset was nearly 8 GB, larger than our team’s laptop could reasonably handle for analysis and modeling, so we took a stratified sample of the data that represented the exact proportion of positive and negative reviews in the full dataset, taking the dataset from n = 6,686,581 reviews to n = 1,337,125, choosing the sample size n could yield results with a confidence level of 99% with confidence interval 0.1. 75% of the data was partitioned for training with the additional 25% partitioned for testing - this was also sampled via stratified sampling to ensure consistent proportions of both positive and negative reviews used for training. The distribution of this sample dataset is shown in the figure below.

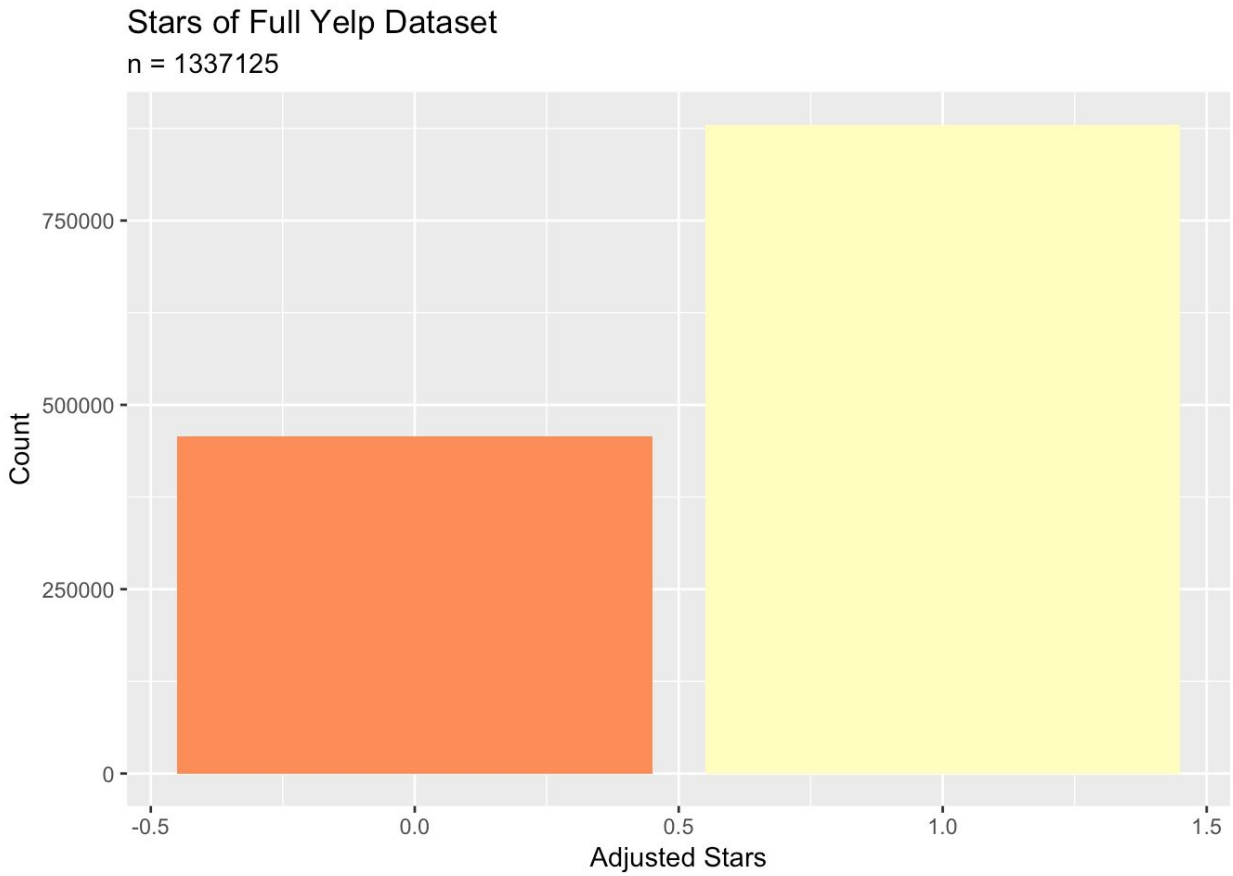


Figure 1. The proportion of positive reviews to negative reviews in our sample dataset, sampled via stratified sampling to match the exact proportion in the full dataset (65.84% positive).

As alluded to in the introduction, we examine three methods to determining the sentiment from this unstructured text. We detail the methodology for these methods individually below:

Model 1: Naive Bayes

As this is, in essence, a supervised probabilistic classification problem, a Naive Bayes model is an appropriate choice for a starting model. The Naive Bayes model calculate the posterior probability using the prior probability, likelihood function, and evidence as described in the following equation:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

This model makes two primary assumptions:

1) The classes are part of a finite and known set, and
2) All features are independent of one another.

Clearly, the first assumption has been met, as all reviews have been split into either positive or negative reviews as detailed in the previous section. The latter assumption, among features, is not fully accurate for this data, but must be simply accepted as an acceptable suit of approximation in exchange for a bit of model error. The features for this analysis will be n-grams, consisting of groups of n-words that appear consecutively in the review text for each review. The n-gram sizes we will consider in this project will be *n* = 1, 2, and 3 - the smaller the n-gram size, the less context we are able to extract from that gram of the review, but the more likely it is to appear in other reviews, a tradeoff we will see the results of in the next section.

To properly perform this analysis, we leveraged the Quanteda package in R - a natural language processing package that allowed us to easily remove common English stop words as well as punctuation when extracting the different n-grams of different sizes. This was done on both the training and testing data exactly the same way to generate document feature matrices for the two. Once generated, the features, or n-grams, were normalized between the train and test sets to ensure model training and testing were done on the same set of features, and, once completed, allowed us to build the model on the training dataset. We then used a built-in function for Naive Bayes in the package as a predict function that, for each test review text, assigns a probability that any provided n-gram is in one of our two classes (positive and negative). We generated an accuracy score, a confusion matrix, and a host of other useful classification statistics to assess model performance.

Model 2: Logistic Regression with Sentiment Score

Being a binary classification problem, a simple logistic regression model is proven effective for determining probabilities of being in one class or the other, but unlike as in Naive Bayes, we take another approach to determining class - sentiment analysis. Using the Python package, TextBlob, a library built on top of the Natural Language Toolkit library, we can feed a sentence as input and receive a tuple containing a polarity and subjectivity score. Rather than using a complicated deep learning model, TextBlob uses an extensive dictionary containing

thousands of “coded” words containing scores based on its grammatical use, parts of speech, and more. It uses this as well as context of negation words as inputs to an averaging formula that returns the score for the input text.

The polarity score ranges from -1.0 to 1.0 and describes how negative / positive the input text is based off of these rules. As a brief example, a review such as “I wish I could give this location 0 stars. It is the worst. There's a 98% chance they'll get your order wrong. NOTHING I ordered today was correct. Avoid this location at all costs!!” receives a polarity score of -0.890, with a more positive review such as “Delicious! Delicious! Delicious! Their pizza is better than Pizzeria Bianco, in my opinion and their nutella crepes are to die for!” receives a much nicer score of 0.906. On the other hand, the subjectivity score ranges from 0.0 to 1.0 and measures whether the input text is more factual or opinion-based, respectively. As both of these values are computed at once, we decided to use both of these models for our model.

To ensure our model was robust, we set up the TextBlob library by downloading all available corpora for determining sentiment scores. Once the installation of the package was complete, we wrote a lightweight Python script to compute the scores for each review text in the sample dataset, both training and testing, and computed the logistic regression model in R, feeding in polarity and subjectivity scores to assess whether the review was positive or negative. Although the task seems simple, as negative polarity scores indicate negative reviews and vice versa for positive, many reviews that contained both positive and negative reviews tended to receive a wide-range of polarity scores, making a logistic regression model necessary for computing cut points to weight probabilities of being in one class higher than the other for this middle portion of scores.

Model 3: TF-IDF Ranked K-Nearest Neighbors

The TF-IDF ranking system has become one of the most prevalent and intuitive algorithms in information retrieval since its introduction in 1972 by computer scientist Karen Spärck Jones. The algorism works taking in a corpus of texts and finding weighted-term frequency for each document as well as an inverse document frequency score for each term, computed with the following equations below:

$$\textit{term frequency } (tf_{t,d}) = \sum_{t \in d} \textit{for all terms t in document d}$$

$$\textit{weighted term frequency } (w_{t,d}) = 1 + log_{10} \bullet tf_{t,d}$$

$$\textit{inverse document frequency } (idf_t) = log_{10} \bullet (\frac{N}{df_t})$$

$$tfidf\ score = \sum_{t \in d} w_{t,d} \bullet idf_t,$$

The score has proven so successful since it appropriately weights how often a term appears in a document (indicating importance for the review) as well as how rare the term is in the corpus of documents (also indicating its importance). Considering the latter review exemplified in the previous section, the term “Delicious” is stated three times, making its weighted term frequency weighting higher than most other terms in determining the tf-idf score

for the document, however, if the term were common in many reviews, then this importance would be reduced due to a lower inverse document frequency weight.

When a “query” text is inputted into the model, a bit of preprocessing must occur before computing any score: the query is tokenized and each token is stripped of its punctuation, capitalizations, and numerals before being run through a stemming and lemmatization algorithm through the Natural Language Toolkit library in Python, which normalizes all the terms in the query to match the same normalization applied to the training reviews. From here, stop words are removed from the query, as these words tend not to provide any additional information, and the tokens are merged to reform a sentence as before. the tf-idf score is computed and summed for all terms in the query as well as all terms in every document in the training corpus, which then produce a series tf-idf vectors for the query terms. With these, we are able to compute cosine similarity between each of the vectors to determine similarity between documents and the “query” input, which we can then select the top-k of and perform k-nearest neighbors to determine the majority class between neighbors. Other than in the 2014 paper, “KNN with TF-IDF Based Framework for Text Categorization” written by Bruno Trstenjak, Sasa Mikac, Dzenana Donko, this model has not been evaluated in other settings, making this an experimental model we will have to code ourselves from scratch in order to implement.

Due to its ability to process text quicker than R, we use Python to implement the TF-IDF ranked k-nearest neighbors algorithm. Due to the nature of the algorithm having a local model and thus having to evaluate nearly every training point for each test point, we sought cloud-computing resources through the Salamander cloud computing provider, which supplied us with a dedicated Amazon Web Services server with 4x vCPU and 61GB RAM, which significantly sped up computation time for this algorithm.

Analysis and Results

Model 1: Naive Bayes

While this was the simplest model to implement out of all the ones tested in this project, it required a considerable amount of computing power to run, taking several hours to train a model with a specific n-gram limit. However, this seemingly simple model turned out to be a highly accurate one with all of the n-grams tested, progressively achieving higher accuracy at rate 85.95%, 89.42%, and 90.30%. While this increase in accuracy was gladly welcomed, it came at the cost of increased computing time with each addition to the n-gram value.

Model 2: Logistic Regression with Sentiment Score

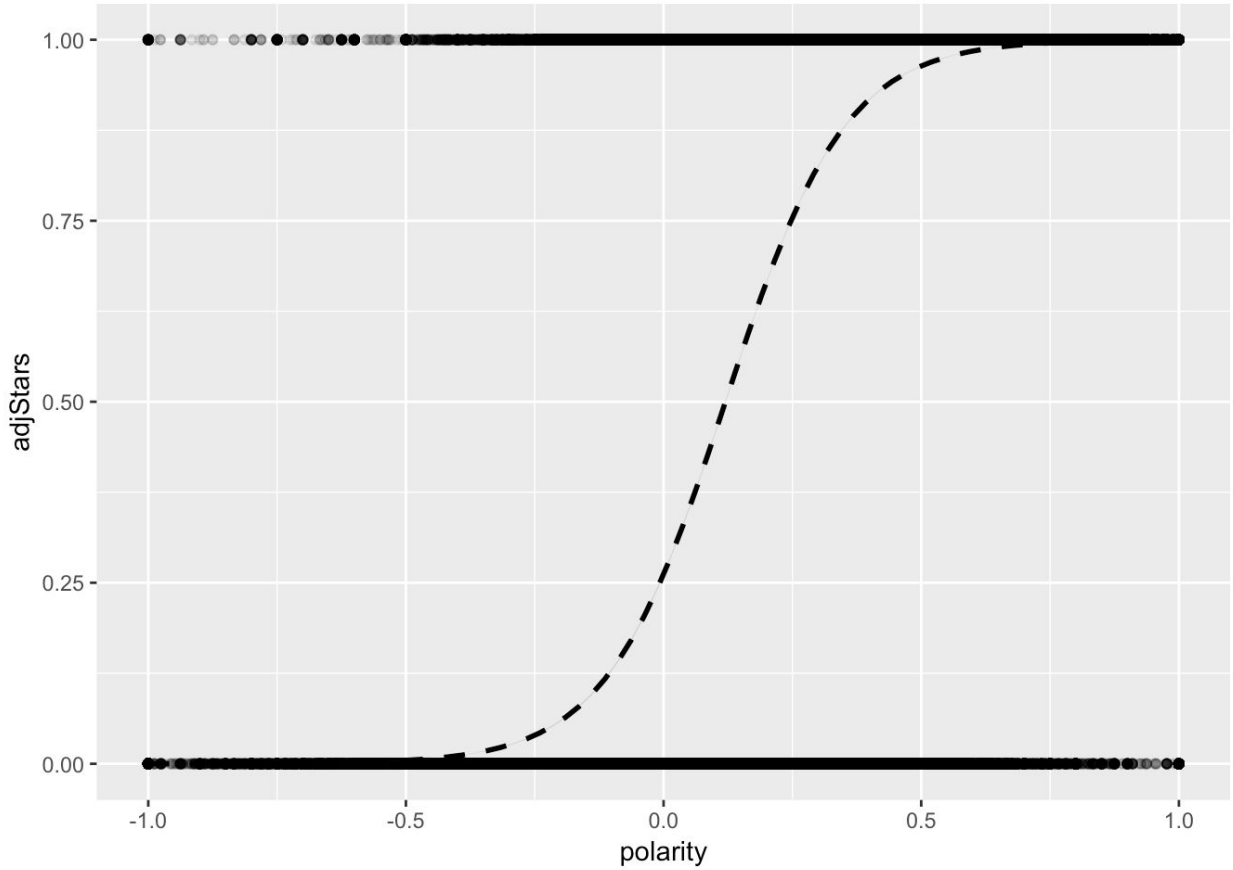
Computing the sentiment score for both the sample training and testing datasets was relatively quick given its large size - in total, polarity and subjectivity computations took around 6 hours of computation time. With this being such a simple logistic regression model to run with only two explanatory parameters being provided to predict a binary class, the actual model computation ran in a matter of seconds.

As a form of variable selection to ensure that both polarity *and* subjectivity were significant variables in producing an accurate prediction for class, we used lasso before modeling with minimum value for lambda being $\lambda = 0.0017$ determined through cross-validation. Even with this small penalty factor, lasso resulted in the subjectivity parameter being completely

stripped from the model, instead leaving the intercept and polarity parameter as the only terms in the model to predict class. The model took the form:

$$adjStars = \frac{e^{-0.9300163 + 8.0327894 \cdot polarity}}{1 + e^{-0.9300163 + 8.0327894 \cdot polarity}}$$

Below, we plot the result of this logistic regression model, showing the surprisingly muddled polarity scores between the two classes - with a cut point between classes appearing around a polarity score of 0.12.



Despite this, the model performed well with a prediction accuracy of 79.2% on the test dataset. Interestingly, running the model using polarity as the only variable in predicting class, the prediction accuracy on the test dataset increases to 80.8% with the model

$$adjStars = \frac{e^{-1.036802 + 8.637981 \cdot polarity}}{1 + e^{-1.036802 + 8.637981 \cdot polarity}}$$

variating slightly from the model above resulting from lasso shrinkage.

Model 3: TF-IDF Ranked K-Nearest Neighbors

Even with the sample size being significantly reduced from the full dataset, the computations for this model immediately became a downside of this model, as building the index for the model took over 3 hours and computing results for the test dataset through the

Salamander cloud computing service took over a week of calculations, making this model not scalable in its current form. Even when implementing a more efficient approximation of the ranking system using a Champion list, which precomputes, for each term t in the collection, the set of the r documents with the highest weights for the term (fixing the value of r in advance to equal the square root of the number of sample training reviews), the computation time still took several days to see completion. For this reason alone, unless the training and testing corpus are relatively smaller, the computing resources are significantly buffed, or the algorithm is written to be parallelized (which is possible but not implemented due to time), it is *not* recommended for the purposes of text classification.

Since computing the top *k* documents is equal in computation time to computing the top *k + 1* review, ties for even number of *k* were broken by considering the class of the *k + 1* review, as there are only two classes and this guarantees a tie breaker each run.

As the number of reviews for each neighborhood was flexible, we computed the results for a set number of *k* values, including *k* = 1, 5, 10, 15, and 30 reviews to consider in a neighborhood. The highest accuracy value obtained from the dataset stemmed from *k* = 30, which was an prediction accuracy of 77.8%.

Overall Analysis and Results

Of the three models we ran, in terms of time complexity for the sample dataset, Naive Bayes ran the fastest, with logistic regression using sentiment analysis scores a close second, and TF-IDF ranked k-nearest neighbors dragging in at a very, very slow third. The table below summarizes the accuracy of the various models tested on the sample testing dataset, which represents a confidence level of 99% and confidence interval of 0.1 from the population dataset.

Model	Accuracy
Simply picking the majority class (adjStars = 1)	65.8%
Naive Bayes (1-gram)	86.0%
Naive Bayes (2-gram)	89.4%
Naive Bayes (3-gram)	**90.3%**
Logistic regression with sentiment analysis scores	80.8%
Logistic regression with sentiment analysis scores (lasso)	79.2%
TF-IDF ranked k-nearest neighbors (k = 1)	72.0%
TF-IDF ranked k-nearest neighbors (k = 5)	76.7%
TF-IDF ranked k-nearest neighbors (k = 10)	77.6%
TF-IDF ranked k-nearest neighbors (k = 15)	77.8%

TF-IDF ranked k-nearest neighbors (k = 30)	77.9%

Based on these results, the optimal model we would recommend for the purpose of this problem statement of text classification in the context of reviews would be a Naive Bayes model using 3-grams, which also happens to be the simplest model to construct of the three tested, taking the least amount of computing time as well.

Conclusions and Remarks

From the above section, it is clear that based on these testing results using the Yelp dataset, the best model performance originated from the **Naive Bayes model using 3-grams chunks of text** to assess class probabilities. However, it is interesting to note that *all* the Naive Bayes models outperformed both the logistic regression using sentiment analysis scores and the TF-IDF ranked k-nearest neighbor models by more than 5 points of accuracy. This was rather surprising to us, as Naive Bayes is arguably the less-complex model of the three and simplest to compute for, yet, this shows that for tasks that might seem very complicated to complete well, simpler models should *not* be ruled out for their simplicity alone.

Of the three models used throughout this project, Naive Bayes has the strongest grounding in statistics, with sentiment analysis and TF-IDF ranked more in information retrieval, computing, and generalization. Bayesian probability is a well-studied and understood field, meaning that the Naive Bayes model is a tried and true statistical approach for a variety of problems.

Future work on this project would continue to explore classification models that have more roots within statistics, regardless of simplicity, as well as note the Naive Bayes model performance in this context with a wider variety of n-gram values to further observe the tradeoff between context, document similarity, and computing performance for prediction power.

References

https://arxiv.org/pdf/1605.05362.pdf
http://cs229.stanford.edu/proj2014/Chen%20Li,%20Jin%20Zhang,%20Prediction%20of%20Yelp%20Review%20Star%20Rating%20using%20Sentiment%20Analysis.pdf
https://pdfs.semanticscholar.org/9c85/836ffaa9dfb3523b793f0d41198d13621b6a.pdf
http://zhongyaonan.com/predict-rating-of-yelp-user-review-text/
https://planspace.org/20150607-textblob_sentiment/
https://textblob.readthedocs.io/en/dev/
https://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2013/178.pdf
https://salamander.ai
https://www.yelp.com/dataset/challenge