Negativity in positivity: What undermines a good restaurant?

Kevin Bowden KKBOWDEN@UCSC.EDU
Johnnie Chang CCHAN60@UCSC.EDU
Robert Chen HCHEN118@UCSC.EDU
Bernardo Gonzalez BEAUGONZ@UCSC.EDU

Project Repository: https://github.com/JohnnieDM/CMPS242Project

1. Introduction

Yelp is a crowd-sourcing website where users share and read about their experiences with businesses in the form of reviews, which consist of 5-star evaluation and textual descriptions. While the star giving system provides an efficient way to roughly summarize the rating of a business, very often the important details about the experiences of the users are within the textual descriptions. In particular, we have noticed that sometimes a business is described to have certain flaws, even though on average they receive a high number of stars. For example, given that a restaurant has a 4-star average, if we go through the low-rating reviews, we may find that though the food is tasty and staff are friendly, some customers have experienced unsatisfactory sanitariness issues. If such issue is a recurring topic in the reviews, it would be quite likely that the restaurant has a systematic problem with hygiene. We think that such negative knowledge helps users gather a better understanding of the business. Our system will provide key phrases that negatively describe each business, which will be a similarly efficient way for users to filter out businesses that are not of their interest.

2. Problem Statement

We want to learn key phrases that negatively describe businesses, using the textual descriptions and star evaluations in the reviews provided in the Yelp data set. This negative tags will be a efficient way for users to filter out businesses that are not of their interest. Contrary to the common approaches seeking positivity in the data, the goal of our learning problem is to generate negativity tags, which are the issues that the businesses may have. Our work was motivated by (*Sentiment Analysis of Restaurant Reviews*, n.d.). To address this problem, we will train a Naive Bayes classifier using the features extracted from the process mentioned below. We then utilize the most discriminative words learned in the sentiment classification phase and extract the key words or phrases from them to provide us with the most negative reviews.

3. Feature Engineering

We extract the text reviews from our corpus and automatically annotate them based on the star rating the user submitted; a 2 or less was deemed a "negative" review while 3 and up is "positive". Our corpus has over 2 million reviews, and the feature extraction and prediction steps in our pipeline become much more expensive as our training data increases in size. For early development we randomly sampled 1 percent of the dataset for our train set and a different 1 percent for our test set. We did experiment with different percentages of the original dataset, this is further discussed in our results section.

3.1. N-grams

N-grams refer to a "gram" consisting of N members, commonly in natural language processing N is refined to represent the number of words in a given "gram". A general process of N-gram generation is to first tokenize the text, then perform refinement over the tokens via various pruning measures before finally creating our N-gram representation. When using N-grams one of the most important decisions is the value of N to use. This value can greatly vary based on the language of the input text, domain knowledge, and the systems task(Raschka, 2014). As N increases, so does the number of features and the runtime of our feature generation module. Since we have a lot of data, we kept N relatively low and focused on Unigrams (N=1) and Bigrams (N=2).

3.1.1. UNIGRAMS

To generate the unigrams, we preprocessed the text of the reviews by first tokenizing our input into each individual word and converting all of them to lowercase; allowing our classifier to treat words like *"Machine"*, *"machine"* and *"MACHINE"* as the same *"machine"* feature. Our second stage of processing removed numbers, punctuation and

stopwords from the text, leaving us with our unigram features.

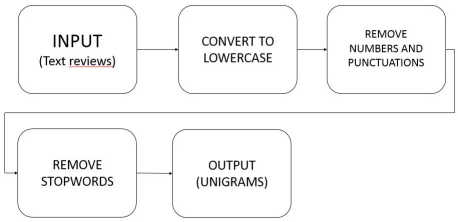

Figure 1. Unigram features

3.1.2. BIGRAMS

To generate our Bigrams we used two layers of tokenization, sentence based and word based. This is important as it avoids the last word in a sentence from being paired as a bigram with the first word of the proceeding sentence. We used the same pruning methodologies as with our Unigram features in order to reduce the feature space and create our bigram features.

3.2. LIWC

We also made use of the feature space provided by LIWC, the Linguistic Inquiry and Word Count framework. LIWC provides us with a diverse set of features, ranging from pragmatic surface text analysis such as part of speech tagging, to more advanced psychometric properties which represent varying degrees of emotion within the text (James W. Pennebaker & Booth, 2007). When more closely examining the different feature options we determined that for the sake of this project, we would focus on sentiment classification by only using those features directly related to positive and negative emotion. We will also point out that LIWC is not an open source software and for our purposes we made use of a script authored in the NLDS lab at UCSC which simulates the LIWC2007 functionality very closely.

3.3. TF-IDF

One further metric we are interested in testing is a comparison between a baseline frequency value for our features to a value generated by the term frequency-inverse document frequency, TF-IDF. Previous work has seen that matching with weighted terms, as with TF-IDF, out performs matching done with just the term values(Karen S, 1972). We are interested to see if this trend will present itself within our project. To compute this we first calculate the term frequency

$$TF(t) = \frac{\#tindocument}{\#termsindocument}$$

Now we calculate the inverse document frequency

$$IDF(t) = \log \exp \frac{\#documents}{\#documentsthathavet}$$

Thus, TF-IDF is the product of these two values.

$$TF - IDF(t) = TF(t) \cdot IDF(t)$$

Our baseline frequency value is calculated from the frequency of a term with respect to all of the terms we have seen.

4. Model Formulation and Methodology

We extracted the *text*, *categories*, *stars*, *business_id*, and *name* fields from the associated json and mixed pandas with regex to wrangle our data from json into a more condensed csv output. This csv was then further processed in our feature extraction phase, generating pickle files which could then be directly loaded as input to our Naive Bayes classifier. We chose to generate and load from these pickle files to decrease our training time and to maintain the same training set between different iterations, enforcing consistency despite our use of random sampling. All of our modules were written in Python 2.7.

The Naive Bayes classifier that we wrote is based on the algorithm we had explored in class which makes use of the following decision rule:

$$lnp(x|C_{pos}) + lnp(C_{pos}) > lnp(x|C_{neg}) + lnp(C_{neg})$$

Where

$$P(x|C_k) = \prod_{i=1}^{D} \mu_{k_i}^{x_i} (1 - \mu_{k_i})^{(1-x_i)}$$

With the domain of k being either pos or neg for our task. We note that we are working in the log space in order to avoid underflow issues.

We decided to use a Naive Bayes classifier because of its high accuracy and ease of implementation. Moreover, we discussed in class the effectiveness of Naive Bayes at handling NLP/text classification tasks, making it an ideal candidate for our project. Further, the training of the Naive Bayes classifier relies on the frequency of each feature, this

model suits our approach perfectly; we use this frequency to determine the most discriminative words learned by the classifier.

We apply laplace smoothing to our classifier as our previous experimentation indicated this would yield better results.

$$P(w_i|c) = \frac{freq(w_i, c) + 1}{\sum_{w \in Vocab}(freq(w_i, c)) + |Vocab|}$$

Figure 2. Model

5. Evaluation

We are primarily interested in evaluating our system in three different ways.

Our Naive Bayes classifiers ability to pick out negative words with different features enabled.

A three way comparison between Naive Bayes, Logistic Regression, and SVM classification.

Comparing our accuracy using baseline frequency values vs TD-IDF values.

To evaluate the effectiveness of our system to predict labels we calculated the accuracy. We realized that just including the accuracy of our system can be misleading (Steven Bird & Loper, 2014), so we also analyze the precision, recall, and F-measure of our approach using the features shown in figure **??**. In this stage we can look also at the discriminative features for our classifier to see if they logically make sense. For our second stage of evaluation, we compare our results to the results obtained using others classification techniques, specifically SVM and Logistic Regression. As we used these models only to compare with our results, we didn't implemented them from scratch but instead used the precooked packages from the scikit-learn python package. Lastly, we observed which feature set returned the highest levels of success and retested that data using TD-IDF values instead of the baseline frequency values. Results for theses evaluations are shown and discussed in the next section.

6. Results

The results obtained are shown in tables 7, 7, and 7. Table 7 shows the results using different features for the same model (Naive Bayes). Due to the lack of space in the table, we denote UNIGRAM and BIGRAMS with U and B, respectively. Tables 7 and 7 show the results using different models, specifically, SVM and Logistic Regression.

Some of our results were expected, while some were surprising. When looking at the quality scores we received, consistently the model trained only on Unigrams had the highest F-score, we were expecting the model with unigrams + LIWC to score the highest at this stage since the LIWC features were meant to detect positive and negative emotion. We also note that the model trained on both unigrams and bigrams scored lowest, this is a result that we did expect. We expect that with the bigram features, since there is a significant increase in the feature space (about 180k bigrams compared to 26k unigrams), there is bound to be many bigrams that are non-informative. Our feature space in this case is to dense, and as a result the accuracy suffers. When we observed the most discriminating words, looking at the unigram model trained on 5% of the data, we also notice something unexpected. Of the top 1000 discriminative words, over 90% of them overlap. Meaning they have a very high frequency for both positive and negative. While some of the discriminative words were also reasonable "negative" words, such as "fee, pain, break, barely", there were a lot of words that seemed to us to be out of place, such as "juice, funny, resort, lounge". We predict this is a result of (1) not using enough data and (2) not categorizing the businesses before training our models and classifying new data. In the future, we realize the business type should be accounted for when training the data.

For our three way comparison between our model and scikit-learns Logistic Regression/SVM model we received satisfactory results. Our Naive Bayes in it's best performance does drop 2-3% behind the performance of the precooked models. However, we feel satisfied that we received such a close F-score. We believe that our model performed worse because while LR and SVM can rely on the data points directly as parameters, our relies more on the features individually. Our results seem to be in line with what was discussed in class where given a lot of data, it is possible for SVM and LR to do better than Naive Bayes. We will point out, however, that Naive Bayes ran *significantly* faster than either of the other two models. Essentially, you trade off a lot of run-time for a 2-3% drop in accuracy.

Our third hypothesis predicted the TF-IDF frequency value

would outperform the baseline frequency value. What we see in our actual data output however is a very close race between using unigrams with the baseline frequency and unigrams with TF-IDF. We are satisfied that the results are so close, and predict that given more refined feature generation, we might see TF-IDF pull ahead of the baseline.

We did attempt to test the scalability of our code by running our system with larger percentages of the data. We noticed that our F-score remained fairly consistent between 1%, 5%, and 10% of the data being used to train our model. We did not test using the entire data set, the most we were able to test was with 80% of the data with just unigrams enabled and we still experienced various performance issues due to hardware restrictions.

7. Future Work

There are a few directions this work could branch of to in the future. The largest issue that we feel we did not adequately address was the amount of data we did our comparisons with. Given more time, we would make further optimization to our framework such that it can handle the large data payload more efficiently. As we have seen in this project, N-grams are a very powerful and simplistic feature. Trying to use higher order N-grams could also yield interesting results provided the features are added cautiously such that you don't blow up your feature space with non-informative features. This leads us to the third, and possibly the most important item on our future work list - more advanced feature engineering. Throughout the development of this code we have struggled with finding the optimal set of features. If work were to continue on this project, we believe the largest area for improvement would be the feature engineering.

DATA	FEATURES	ACCURACY	PRECISION	RECALL	F-MEASURE
1%	U	75.24%	79.36%	93.01%	85.64%
	U + B	61.02%	79.67%	68.36%	73.58%
	U + L	75.31%	79.42%	93%	85.68%
	U + B + L	63.54%	79.43%	72.98%	76.07%
	U + T	75.43%	79.96%	92.42%	85.74%
5%	U	75.82%	79.31%	94.06%	86.06%
	U + B	70.28%	79.29%	84.64%	81.88%
	U + L	73.11%	79.31%	89.43%	84.07%
	U + B + L	71.41%	79.28%	86.58%	82.77%
	U + T	75.81%	79.58%	93.57%	86.01%
10%	U	75.82%	79.31%	94.06%	86.06%
	U + B	70.28%	79.29%	84.64%	81.88%
	U + L	73.05%	79.20%	89.48%	84.03%
	U + B + L	71.41%	79.28%	86.58%	82.77%
	U + T	75.43%	79.96%	92.42%	85.74%

Table 1. Results using Naive Bayes

DATA	FEATURES	ACCURACY	PRECISION	RECALL	F-MEASURE
1%	U + B	79.36%	79.36%	100%	88.49%
	U + L	79.36%	79.36%	100%	88.49%
	U + B + L	79.36%	79.36%	100%	88.49%
	U + T	77.61%	78.45%	98.55%	87.36%
5%	U	79.43%	79.43%	100%	88.53%
	U + B	79.43%	79.43%	100%	88.53%
	U + L	79.43%	79.43%	100%	88.53%
	U + B + L	79.43%	79.43%	100%	88.53%
	U + T	78.38%	79.5%	98.08%	87.81%
10%	U	79.64%	79.31%	100%	88.66%
	U + B	79.64%	79.31%	100%	88.66%
	U + L	79.64%	79.31%	100%	88.66%
	U + B + L	79.64%	79.31%	100%	88.66%
	U + T	%	79.96%	92.42%	85.74%

Table 2. Results using Logistic Regression

DATA	FEATURES	ACCURACY	PRECISION	RECALL	F-MEASURE
1%	U	79.29%	79.29%	100%	88.45%
	U + B	61.02%	79.67%	68.36%	73.58%
	U + L	75.31%	79.42%	93%	85.68%
	U + B + L	63.54%	79.43%	72.98%	76.07%

Table 3. Results using Support Vector Machine

References

James W. Pennebaker, M. I. A. G., Cindy K. Chung, & Booth, R. J. (2007). The development and psychometric properties of liwc2007..

Karen S, J. (1972). A statistical interpretation of term specificity and its application in retrieval..

Raschka, S. (2014). *Naive bayes and text classification i - introduction and theory.*

Sentiment analysis of restaurant reviews. (n.d.).

Steven Bird, E. K., & Loper, E. (2014). *Natural language processing with python.*