Text based rating predictions from beer and wine reviews

Benjamin Braun
Department of Computer Science and Engineering
University of California, San Diego
bebraun@cs.ucsd.edu

Robert Timpe
Department of Computer Science and Engineering
University of California, San Diego
rtimpe@cs.ucsd.edu

ABSTRACT

Predicting ratings from text reviews is a very challenging text mining problem with a multitude of practical applications on online review platforms. This task involves both a sensible feature extraction from the review text as well as creating a model with a high accuracy. For this project, we deployed and compared a number of natural language processing methods for extracting features from three large datasets of wine and beer reviews. We used these features to create and compare the performance of classification and regression models to predict fine grained ratings.

Keywords

Data mining, sentiment analysis, Naive Bayes, Ridge Regression

1. INTRODUCTION

The goal of our project is to predict ratings from review text. Online reviews are an important factor for users to decide between products. They are extensively used for movies, on online shopping sites, restaurant reviews and food critique platforms. Most platforms allow users to submit a written review as well as a numeric label, for example stars or points on some scale.

We tried a number of models to predict ratings from beer and wine reviews, but we eventually settled on linear regression and naive Bayes classification. These models are relatively simple, but often achieve reasonably good performance in practice. In addition, their simplicity allows them to efficiently scale to huge training sets. This was an important criteria for this project because the datasets we use have millions of reviews, so training an inefficient model may not be practical.

2. DATA SETS

To compare the quality of our prediction for different domains, we are using data sets collected from popular beer and wine rating sites.

2.1 CellarTracker

The cellar tracker dataset consists of 2,025,995 wine reviews. Each review has 9 different features: wine name, wine id, wine type, wine year, review score, review time, the user name and user id of the reviewer, and the text of the review. Of these features, we are mostly interested in the reivew score and review text, so this is the focus of our exploratory analysis. Firstly, we are interested in the distribution of ratings. Figure 1 shows this distribution. The scale goes from 50-100 and is described in [1]. Despite the fact 80-85 is considered an average wine, most of the ratings are clustered around 85-95 range. The mean rating is 88.8 with a standard deviation of 4.2.

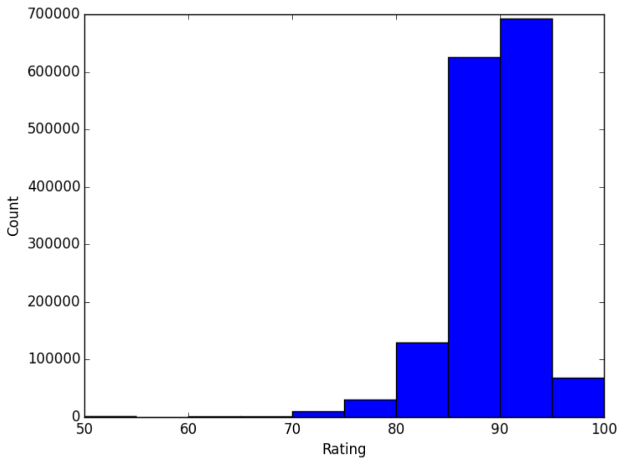

Figure 1: The distribution of wine ratings

As part of our exploratory analysis, we also looked at some properties of the reviews. As figure 2 shows, there is some correlation between review length and wine rating (i.e. longer reviews tend to go with better ratings). We are also interested in some more advanced statistics about the reviews. Figures 3, 4, and 5 show the number of adverbs, adjectives and nouns vs the wine rating. These figures show some correlation between the number of adverbs and adjectives and the rating, and a somewhat weaker correlation between the number of nouns and the rating. We also looked at other parts of speech (such as pronouns) and found very weak correlations. Overall, this shows that we can expect to find some predictive power by looking at review text.

Finally, we also looked at how the ratings varied depending on the time of year. Figure 6 shows the average wine rating by month. There is some small variation, with peaks in the early summer and early winter. This indicates that there may be some difference in the type or quality of wine that people drink in different times of the year that will allow us to make predictions from the reviews.

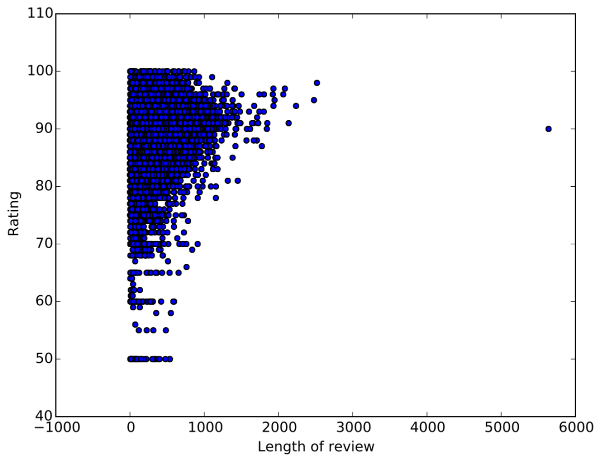

Figure 2: Length of review vs wine rating

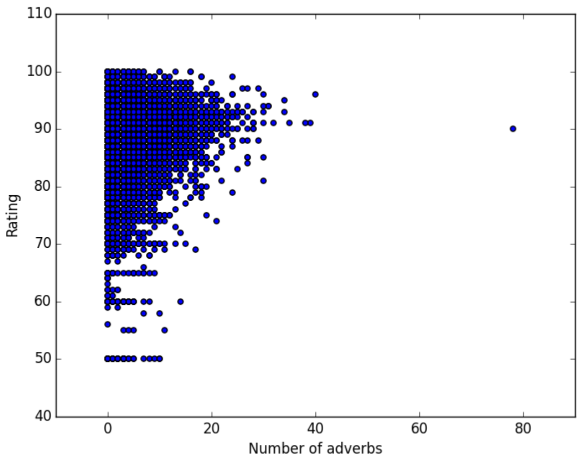

Figure 3: Number of adverbs vs wine rating

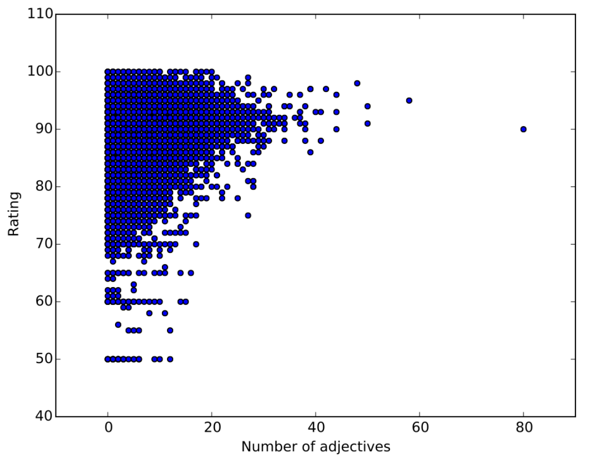

Figure 4: Number of adjectives vs wine rating

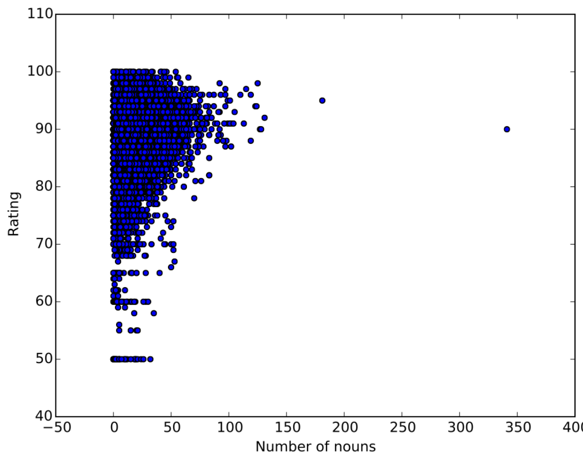

Figure 5: Number of nouns vs wine rating

2.2 BeerAdvocate

The BeerAdvocate data set consists of 1,586,602 beer reviews. Each review has information about the beer: name, style, ABV, id, and brewer id, as well as review information: reviewer name and time, review text, overall rating, and 4 subratings: aroma, palate, taste, and appearance. Figure 7 shows the distribution of overall ratings. The average rating is 3.8 with a standard deviation of 0.7.

We also looked at some of the language properties of this dataset. Figure 9 shows the length of the review vs the overall rating, while figure 8 shows the number of adjectives vs the overall rating. There is a slight correlation between review length and overall rating, with a similar but slightly smaller correlation between number of adjectives and overall rating. We also made similar plots for other parts of speech (omitted for brevity) which showed similar correlations. We expect that this relationship between reviews and ratings will allow us to make predictions about the rating from the review text.

2.3 RateBeer

The RateBeer dataset includes 2,924,127 from 40,213 users which is the whole dataset of the website up to November 2011. Similarly to the BeerAdvocate dataset, most of the reviews also include aspect ratings for the same four aspects. We applied the same analysis to this dataset as for BeerAdvocate. The analysis did not reveal significant differences, so we do not provide any graphs for this dataset.

3. PREDICTIVE TASK

Our main predictive task is to predict review scores from review text. This can be framed as a classification problem by splitting the scores up into ranges. For the wine data, we binned the scores as described in [1]. The BeerAdvocate data has ratings in increments of .5 from 0-5. We binned these ratings by multiplying by 2 to get ratings in the range 0-10 inclusive. The RateBeer ratings are integers in the range 1-20 inclusive. We used these ratings directly as categories.

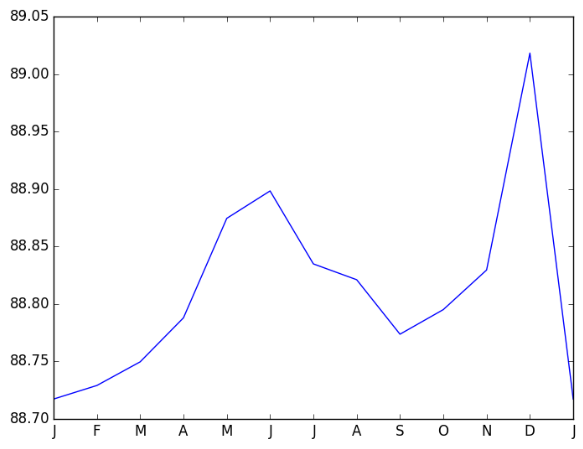

Figure 6: The average wine rating by month

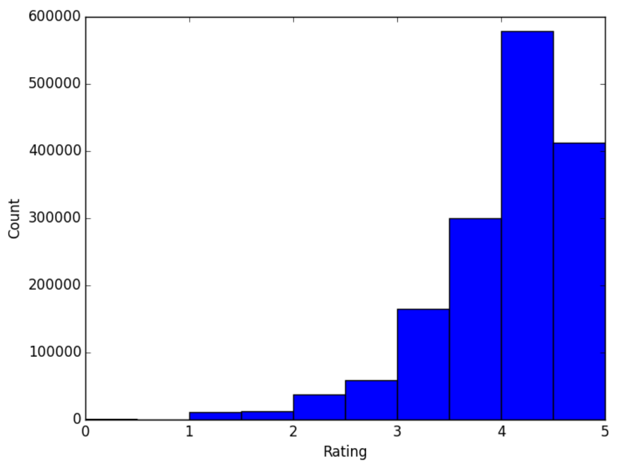

Figure 7: Distribution of BeerAdvocate ratings

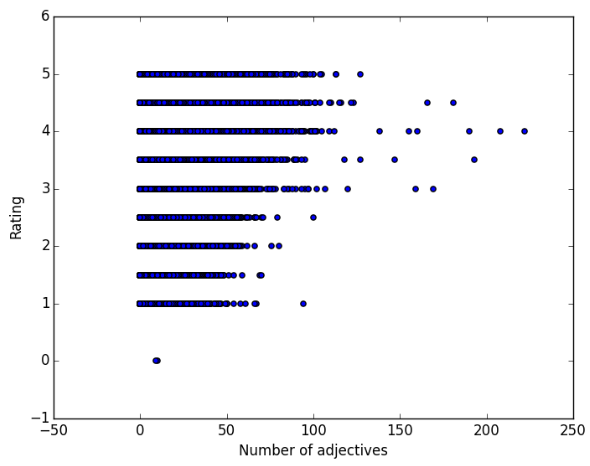

Figure 8: Number of adjectives vs beer advocate rating

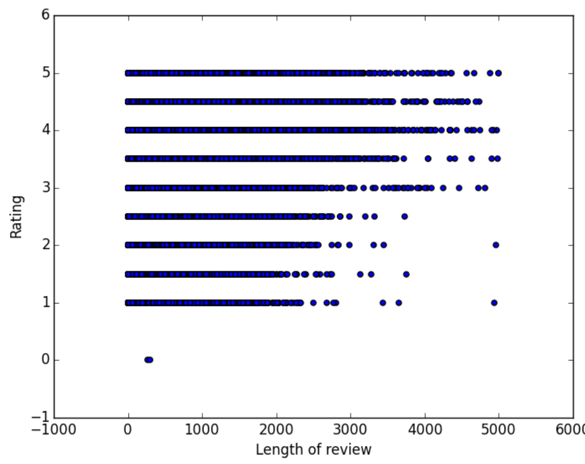

Figure 9: Length of review vs BeerAdvocate overall rating

In addition to the review score, we are also interested in predicting other review properties from the text. We also predict the time of year and beer style from the review text. We didn't try to predict wine style because while the CellarTracker data has the wine variant, it is too fine grained to be used for classification and we had no good way of translating it into more general categories.

3.1 Evaluation of model

We evaluate our model in several ways. In addition to classification accuracy, we also look at confusion matrices to get a more detailed picture of what kind of classification errors our model makes. For regression, we also translate the continuous prediction to a classification based on the previously introduced bins. Additionally we calculate the mean squared value when comparing results from the data set. To compare results with different variance, we also calculate the coefficient of determination R^2.

After preprocessing our data, we randomly shuffle the reviews for each of the datasets and use 90% of the reviews for our training set. The remaining 10% are never used during the training of the model unknows or the hyperparameters and exclusively for the final testing.

3.2 Relevant baselines

The simplest baseline for each of these predictive tasks is a model that always predicts the most common label. The frequency of each label is shown in figure 10 for the CellarTracker data and in figure 11 for the BeerAdvocate data. The two most common labels in the RateBeer dataset are 14 and 15, with frequencies 14.8 and 14.4, respectively (the full table is omitted for brevity).

For regression, the baseline is just the mean value of the data. This would equal an R^2 of zero.

Unfortunately, our results were not as good as similar results from the related work described in Section 4.

3.3 Assessing validity of the predictions

To asses the validity of our model, we will compare our prediction accuracy to the baselines. Additionally, we will use the trained features of our model for qualitative analysis.

4. RELATED WORK

The RateBeer and BeerAdvocate datasets have been used previously in a number of papers. In [8], this data, among other datasets, is used to learn attitudes and attributes by taking advantage of the separate ratings for multiple aspects that are present in the dataset. They introduce a new model to that identifies the described aspect and sentiment per sentence in the review. In a first step, the model identifies sentences that describe a particular aspect out of the rated aspects of a product by learning sentiment neutral aspect words. For these aspects, a rating is predicted using aspect specific sentiment words. This outperforms previous approaches while being scalable to large datasets.

In [9], similar datasets, including the RateBeer, BeerAdvocate and CellarTracker datasets were used for a recommender system that also takes into account the current experience level of a user. They use a latent-factor model with parameters evolving independently for each user to model the individual progress of a user. This outperforms models that just include temporal changes for the community as whole.

Ganu et al. [5] also try to identify sentiment from free-form text reviews to improve recommendation accuracy. They manually annotate a set of sentences from restaurant reviews with category and sentiment labels to train a SVM to categorize sentences int the dataset. They then learn weights for each (category, sentiment) tuple using regression. Their regression-based text ratings are not very accurate, but they are still able to improve recommendations to users with the text compared to using just the numerical rating.

Qu et al. [12] also address the problem of predicting numeric ratings from a user's product review text. They argue that unigrams are unable to copture more complex expressions, for example negation, while N-grams of words occur too infrequently in most training corpuses which lead to extremely sparse vectors that do not work well for predictions. Instead they map sentences, for example *Not particularly amazing*, to opinion triples, containing an opinion root, modifiers and negation words. They use a subjectivity lexicon to identify opinion roots and negation words and learn weights for the opinions triples using a constrained ridge regression. Because their opinion triples are domain independent, they combine this with a unigram model from a domain specific corpus, also trained using ridge regression. They evaluate this on Amazon reviews from multiple categories and show that their opinion model improves the mean squared error compared to n-gram models.

Category	Frequency
50-69	0.31
70-79	2.55
80-85	13.07
86-89	35.37
90-93	40.62
94-97	7.54
98-100	0.54

Figure 10: The frequency of each wine rating category

Rating	Frequency
0	0.00044
0.5	0.0
1	0.69
1.5	0.82
2	2.41
2.5	3.69
3	10.44
3.5	19.02
4	36.73
4.5	20.45
5	5.75

Figure 11: The frequency of each BeerAdvocate overall rating

In [6] Joshi et al. describe how they used text reviews of movies to predict opening weekend revenues. They extract three types of text features from the movie's critic reviews: n-grams, parts-of-speech n-grams, and dependency ralations using the Stanford parts-of-speech tagger and parser. The predictions obtained from a linear regression based on text features are similar to predictions from metadata alone.

In [7] Kibriya et al. describe multinomial naive Bayes and some modifications to improve its performance.

5. FEATURES

Most of the reviews in the datasets we used contain a more or less elaborate text review of varying length. We focus our work on features that can be obtained from the review text as this is available across all review platforms.

Before anything else, we do some preprocessing to clean up the review text. We noticed that many of the wine reviews had some HTML tags in them. We decided to strip these tags so we would be dealing with pure text. However, it would be interesting, as an area of future work, to try and incorporate these tags into a model. Many of the tags are unlikely to have predictive value, but some (such as bold and italics tags) may provide useful information.

While stripping HTML tags removed some noise from the data, some of the reviews are still difficult to parse. For example, there are many examples of words joined by punctuation (and especially ellipsis) that our parser is unable to handle. Unfortunately, cleaning up this kind of data would be a very involved process, so we left them alone.

From the review text we extract the number of words in the review as a feature. To get more accurate word tokens from the text, we use the Python natural language processing library spaCy [3] to tokenize the text into words. Additionally, we use stemming on the words to condense the dictionary by identifying the root words. spaCy uses the WordNet lexical database [4] for this. For example, the phrase *don't* will results in the tokens *do* and *not*. We expect this to significantly reduce the dictionary size, so that we get lower dimensional features for more accurate predictions.

We also obtain a part of speech classification for each of the tokens, e.g. classifying the words into 17 categories, e.g. NOUN, VERB, ADV,... The classification is based on the Google part-of-speech tag set [11].

From the word stems, we generate a feature vector for each review using the bag-of-words model. We limit the dictionary to the 200.000 words provided by spaCy. This eliminates misspelled words and keeps our feature vector relatively small to avoid overfitting. To also capture more complex phrases in the reviews, we also generate bigrams from the tokenized review corpus and select the 500 most frequent bigrams as additional features.

Stainingly dark, coating the glass. Nose and taste of graphite, iron, minerals. Brooding and tannic, with good balance. Quite distinctive. We will look forward to opening our second, and last, in two years.

'dark', 'coat', 'glass', 'taste', 'graphite', 'iron', 'mineral', 'brood', 'tannic', 'good', 'balance', 'distinctive', 'will', 'look', 'open', 'second', 'last', 'year'

Figure 12: An example of a wine rating before and after processing

Because we assume that most of the predictive power of our model comes from the adjectives, nouns, verbs, and adverbs in the review text, we also try reducing our bag-of-words feature to only include the words that were tagged as these parts of speech. Figure 12 illustrates this on an example of a wine rating. This should provide a very similar effect as filtering out stop words, a common optimization technique to make the training faster. We will evaluate the effectiveness of this technique in Section 7.4.

To make computation on these high dimensional vectors more efficient, especially considering the large size of our datasets, we take advantage of the fact that most reviews are relatively short and therefore result in a highly sparse matrix. We convert our feature vectors to a compressed row storage matrix. This provides significant performance improvements when training a linear regression model or a support vector machine.

In addition to features generated directly from the review text, we also extract the current the experience level of a user by determining the number of reviews the reviewer has written so far.

We assume that the language in the reviews will differ between the different beer or wine variants that are being reviewed For this, we categorize the beer reviews into reviews for three different beer styles: lager, ipa and stout. We do this by adding beers that contain the word *lager*, *ipa* or *stout* in the field *beer/style* to the respective category. Other beer styles are ignored. While we might miss some of the beers in these categories, this should mostly avoid falsely categorized beer styles.

6. PREDICTION MODEL

6.1 Naive Bayes

One of the models we used was a naive Bayes model. We started with the simplest implementation of a multinomial naive Bayes model [7]. That is, we measured the prior probability of each class and estimated the posterior distribution of words over each class using Laplace smoothing. We then represented documents using a bag of words model and made predictions based on the estimated parameters.

In addition to the basic naive Bayes model, we also tried a number of improvements. One method we tried was to shrink the vocabulary by ignoring words that appear with approximately the same frequency in each class. The idea is that these types of words will have little to no predictive power because they are equally common in each type of review. To do this, we first measured the frequency with which a word appeared in each different class. We then computed the mean and standard deviation of these frequencies for each word and removed words where none of the frequencies differed by λ times the standard deviation. λ is a hyperparameter that we set using some of the training data as a validation set.

We also tried naive Bayes with a bigram instead of unigram model. So instead of posterior distributions over individual words, we had posterior distributions over each neighboring pair of words in the training set. We hoped that this extra context would improve the accuracy of the model.

Finally, we tried stemming using the spaCy library. This can be thought as another way of reducing the voacbulary size. But rather than removing common words, we map words to their root words. The goal is to reduce noise without while still preserving the meaning of words. This is accomplished using the lemma property from the spaCy api [2].

Even with these improvements, the naive Bayes model still has a number of weaknesses. It completely disregards the structure of the text. It also ignores relationships between words and assumes that the features (i.e. word counts) are independent of each other. Unsurprisingly, classification accuracy suffers as a result of this simplifications.

On the other hand, these simplifications allow for generally good performance. Despite working in very highly dimensional space (i.e. the size of the vocabulary), this model achieves good performance. Using this model allowed us to use training sets with millions of data points, which may not have been possible with a more complicated model. Given the large size of the data sets we used, this was a highly desirable property. The only performance issue we had with this model was using the spaCy library. Parsing millions of reviews was a relatively slow operation, even though the model itself is very efficient.

Despite its simplicity, the naive Bayes model is widely used for text classification, which is why we decided to use it. We also wanted to analyze our model to draw interesting conclusions about the data set, as described in section 7. This can be easily accomplished with a naive Bayes model, for example by looking at the posterior distributions, but is not always possible or easy with a more complex model.

6.2 Support vector machine

We also tried to do classification using a multi-class support vector machine with an radial basis function. While SVM is traditionally a binary classifier, they can also be applied to multi-class classification using a one-against-one approach. The problem is that this requires us to train a classifier for each pair of classes, resulting in $O(c^2)$ classifiers. We tried to apply the library implementation from scikit-learn [10]. However, we could not train the SVMs from enough samples to get significant predictions. We assume that our features are too high dimensional, given the number of training samples we have, to train it on our Laptops.

6.3 Linear regression

Another model that we implemented was linear regression. Again we started with the simple model introduced in class

$$X\theta = y$$

with X being our sparse matrix of text features, θ are the weights associated to each words, and y is the vector of predicted ratings.

However, without any regularization, the model has a strong tendency to overfit. To improve this model, we evaluated a number of different regularization techniques. We evaluated an l_1 regularizer that also minimizes the the l_1 norm of the weights in addition to the mean squared error:

$$\arg\min_{\theta} = \frac{1}{\#\text{samples}} \|y - X\theta\|_2^2 + \lambda\|\theta\|_1$$

This leads to a model with a sparce coefficient vector, reducing the number of non-zero weights. This did not perform well for our datasets and effectively only predicted the mean rating.

Alternatively, we also tried l_2 regularizer:

$$\arg\min_{\theta} = \frac{1}{\#\text{samples}} \|y - X\theta\|_2^2 + \lambda\|\theta\|_2^2$$

This method is also known as ridge regression and was proposed in some of the related work we studied, for example in [12], and proved to provide the most reliable prediction while avoiding the issue of overfitting.

A model with both an l_1 and l_2 regularizer also ran into the same issues as with simple l_1 regularlization.

To fit the hyperparameter λ, we did a grid search over a range of parameter values. For this, we split up our training data using $k = 5$-fold cross-validation. While this did not significantly change the model performance compared to the default value of $\lambda = 1$, we did get a minor improvement in accuracy.

One advantage of the ridge regression model is the good performance when training the model. By taking advantage of the sparsity of the input data, we had no issues with memory capacity. We tried multiple solvers and found that an iterative solver gave the best training performance. We trained the model from the whole training set of more than a million reviews. The model does not require any manual labels on the data and can even cope with the fact that some of the reivews are written in other languages than English.

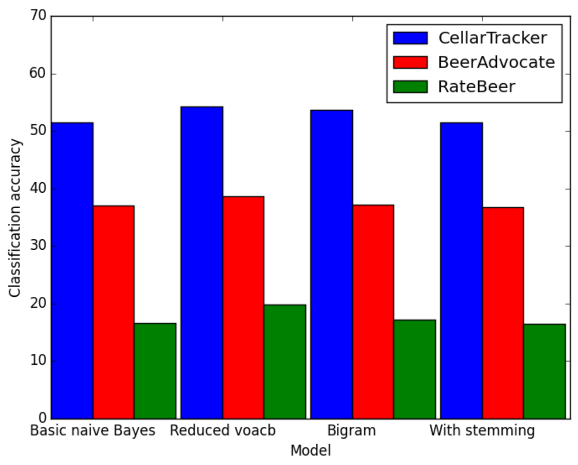

Figure 13: The distribution of wine ratings

The trade-off is that the correlation between features and rating is often not linear, as assumed by the model. Furthermore, our model fails to capture more complex expressions. For example, even with bigrams or trigrams the negation in *Did not get much fruit in front* will not be associated to the fruit. Our model would give this a positive sentiment due to the positive correlation of *fruit*. Due to this, our prediction accuracy is not that good and we are unable to explain all of the variance in the ratings.

A benefit of the ridge regression is that after training the model, we get a feature vector with a positive or negative sentiment rating for each word or bigram.

6.4 Support vector regression

As an alternative to linear regression, we also implemented and tested a support vector regression model. However, we ran into the same issues with this as for the support vector machine for classification. We were not able to train the model from a reasonable amount of samples to get solid predictions.

7. RESULTS

In the following section, we summarize and evaluate the results of our predictions.

7.1 Comparison of text features

We first compare the effectiveness of different methods of feature extraction for both a Naive Bayes and a linear regression model.

7.1.1 Naive Bayes

Figure 13 shows the classification accuracy of the basic naive Bayes model compared to various improvements, on each of the 3 datasets. In each case, the model was able to slightly outperform the simplest baseline described in section 3.2. There was also relatively little difference in performance between each of the improvements (which are described in section 6). Reducing the vocabulary size seems to be the most successful improvement. This makes sense, because reducing the vocabulary size can significantly reduce the amount of noise in the data. Common words like "the", "a", etc. are very unlikely to have much predictive power. But their frequency may easily vary independently of whether the review is positive or negative. This noise can impact the model's predictions, and so one can expect that ignoring these words will improve accuracy. The lack of success of the bigram model was unexpected, but not entirely surprising. After all, two-word pairs do not give much more context than one word pairs. The accuracy of the model with stemming is somewhat more surprising. We expected stemming to be a more significant improvement over the base model.

Figure 19 gives an example confusion matrix for the RateBeer dataset. Looking at this figure, it is clear that the prior probabilty distribution is playing an important role in the model's performance. The prior distribution simply reflects the distribution of ratings, for example as shown in figures 10 and 11. In the case of the RateBeer dataset, most of the ratings are in the range of 10-16, which is also where the model is making most of its predictions (discussed further in section 7.3). However, there is some interesting information in the posterior distribution as well. For example, some of the words with highest probability of appearing in a RateBeer score of 20 (i.e. the highest possible score) are "sweet", "chocolate", "dark" and "best". These are words that one would reasonably expect to be associated with very good reviews. Unfortunately, looking more closely at this list of most probable words reveals other words which are less likely to have good predictive value, such as "a", "beer", and "this". These are likely just words that appear frequently across all types of reviews. The fact that these words have not been removed from the vocabulary represents a failure of the "reduced vocabulary" improvement, which is meant to remove such words. Most likely, these words happened to appear significantly more or less frequently in one or two types of reviews, and so they were not removed. This also explains why this improvement didn't produce a bigger gain in accuracy.

7.1.2 *Ridge Regression*

In Figure 14, we compare the performance of the regression based model by means of different text feature extraction methods based on review score prediction fro the RateBeer dataset. None of the methods significantly outperforms any of the others. The basic bag of words features when using stop words perform the worst with an R^2 of 0.408. Not removing stop words lead to a minor improvement, probably because we did not include a length feature, so the stop words substituted for that without carrying any actual sentiment. Both, stemming and using bigrams lead to further improvements. However, the model is clearly too simple, as it only explains about 40% of the variance in our dataset.

7.2 Domain specific sentiment

The feature vector of the trained regression model associates a weight to each word in the reviews. The weight should be a direct indicator of the sentiment of a word for the specific domain of reviews. In Figure 15 we show the words with the highest weights for wine reviews. Most of the words are domain independently positive like *perfection, awesome,*

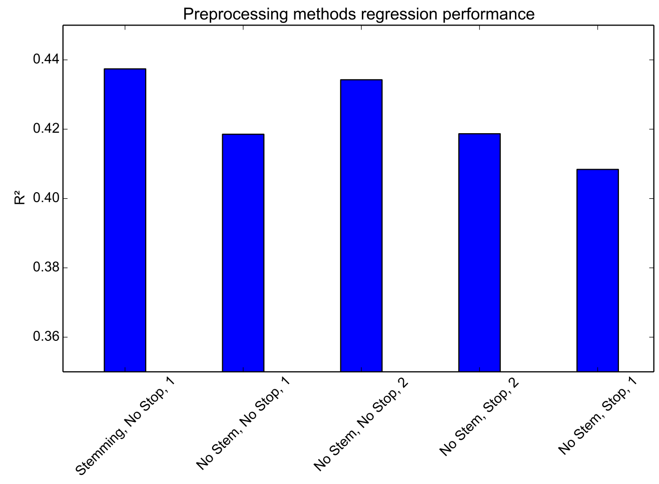

Figure 14: Comparison of different preprocessing methods for regression

magnificent, amazing, while others like *weiner, pritchard* are clearly domain specific (in this case wineyards).

Beer reviews use a clearly distinct language for positive sentiment, as shown in Figure 17. In comparison, the words for negative sentiment are much more similar. As shown in Figure 16 for wine reviews and Figure 18 for beer reviews, both domains seem to agree on a similar language for reviewing bad beers and wines. When we inspected the weights, we also noticed that negative words usually carry a much stronger negative penalty. For example in case of the wine ratings, while *undrinkable* has a weight of -7.13, *perfection* only has a weight of 2.52. A likely explanation is that positive adjectives occur more often with opinion modifiers that relativize the expression, which is not captured by our model.

7.3 Comparison of Naive Bayes and Regression

Our two models had very similar classification performance on each of the 3 datasets. This makes sense because they both represent data in the same way. In each case, the text is represented as a bag of words and relationships between words are ignored. Figure 19 shows the confusion matrix for the naive Bayes model, while figure 21 shows the confusion matrix for the linear regression model, both on the RateBeer dataset. In each case, nearly all of the predicted labels are in the 10-16 range. This makes sense, because approximately 75% of reviews in the training set are in this range. So both of the models are generally accurate, but fail to precisely differentiate between reviews that are very similar (i.e. a 14 vs a 15). This is apparent in the confusion matrices. Both models are making some number of correct predictions, and a high number of incorrect predictions in nearby categories. Figure 20 shows the confusion matrix of linear regression on the CellarTracker data. It shows very similar behavior to the RateBeer dataset, albeit with fewer categories. The other confusion matrices all have similar behavior and are omitted for brevity.

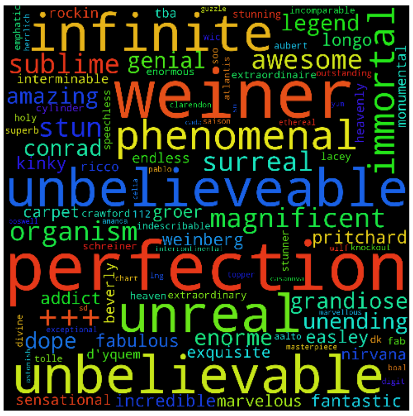

Figure 15: Positively weighted words in wine reviews

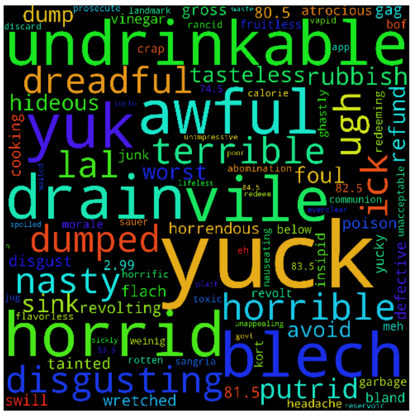

Figure 16: Negatively weighted words in wine reviews

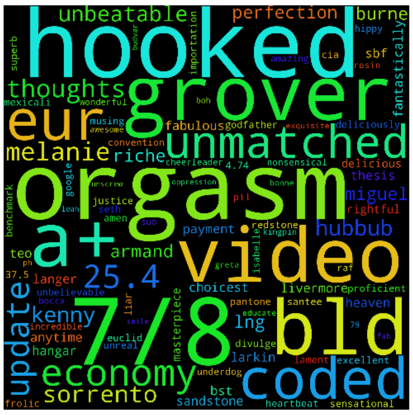

Figure 17: Positively weighted words in beer reviews from RateBeer

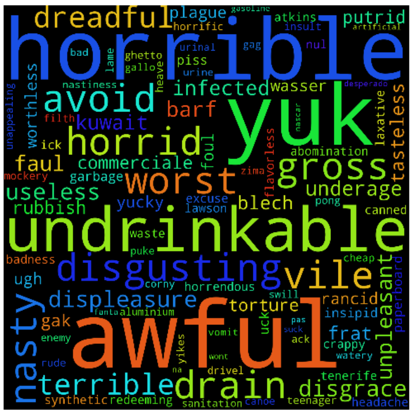

Figure 18: Negatively weighted words in beer reviews from RateBeer

Overall, both of the models are good at predicting approximately how good a review will be, but fail to reliably differentiate between reviews that are very similar. It would also be interesting to try these models on a more evenly balanced dataset. As described in section 2, all of these datasets have far more positive than negative reviews. It would be interesting to see how these models perform on a dataset with an approximately equal number of positive and negative reviews. This would make it harder for the model to simply always predict that a review is good. But this is not how these datasets are distributed, and we wanted to focus on analyzing real data. After all, if most of the reviews fall into a small range of the rating scale, then any real-life application of these models will likely have to differentiate between such reviews.

As shown with the CellarTracker dataset, we can artificially improve accuracy by reducing the number of categories. But the overall behavior of the models doesn't change, so there is no real reason to do so. And again, we wanted to explore the performance of the models in a realistic scenario, which means keeping the same categories as the original data.

7.4 Information content in different parts of speech

We also evaluated which parts of speech were most useful for predicting the review score. To do this we used the previously described parts-of-speech tagger while creating the bag-of-words features. We then filtered out words that did not match our current parts-of-speech filter. As shown in Figure 22, our predictions are most accurate when including when including all of the words. However, limiting the words to only include nouns, adjectives, verbs and numbers did result in nearly the same accuracy. As expected, adjectives are most useful as they carry most of the sentiment. We also noticed that the accuracy of the tagger was not perfect. Especially when filtering out all nouns, adjectives, verbs and numbers, the remaining words with strong weights were nearly exclusively misclassified ones.

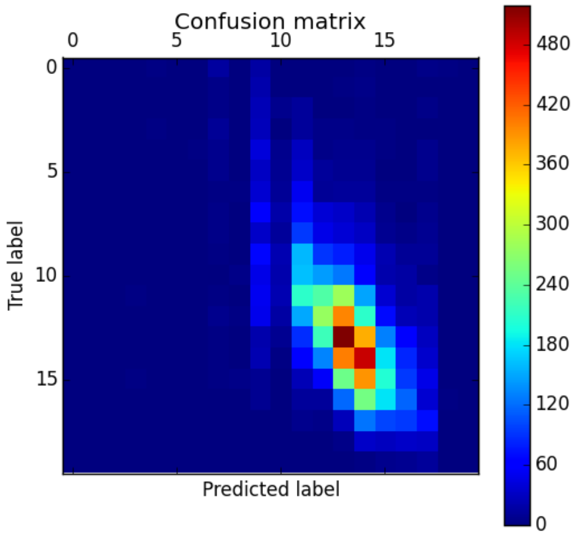

Figure 19: Confusion matrix of RateBeer ratings using naive Bayes with reduced vocabulary

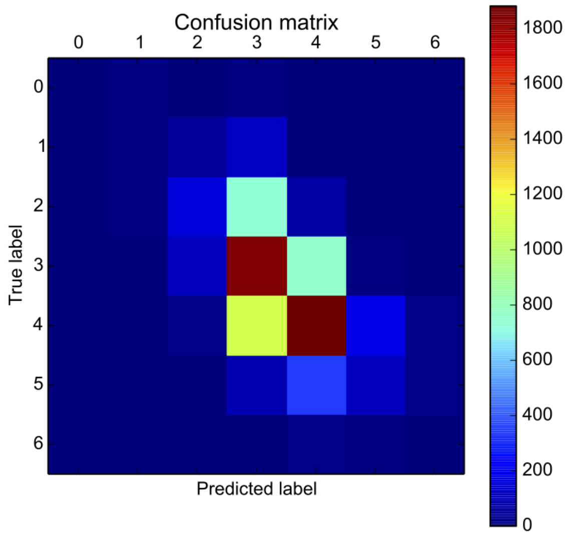

Figure 20: Confusion matrix of wine ratings classified using regression

7.5 Temporal changes in review language

Once we decided on the models, we also applied them to the task of predicting time of year from review text. We split the reviews into 4 categories (Winter, Spring, Summer and Fall) to do classification. We choose to split the reviews by season rather than month in the hope that seasonal trends/differences would be more significant than monthly ones.

The naive Bayes classifier with a reduced vocabulary achieved an accuracy of 29.25% on the wine data. This is only slightly better than always guessing "Spring", which appears 26.58% of the time in the training data. The classifier almost always predicts "Winter" or "Spring" (the two most frequently occurring seasons), which indicates that it is not learning very much about which words predict which seasons (i.e. it is mostly making predictions based on the prior, rather than posterior distributions). Looking at most probable words in the posterior distributions supports this theory. For example, many of the top words in the "Winter" and "Summer" posteriors are the same. This includes words like "and", "the", and "with". There are some differences between these distributions, but there are simply too many non-predictive words with high probability for these differences to be effective in making predictions. A more effective method of reducing vocabulary size and ignoring words like "the" and "and" may result in more accurate prediction.

7.6 Review language for different beer variants

We also used our model to identify the beer variant that is described in the review. For this, we trained a classifier to identify lagers, IPAs and stouts based on the text in the reviews. To make this task more interesting, we filtered these words from the reviews. For training, we had between 200,000 and 300,000 reviews from each beer style. Our classifier had an accuracy of 0.870. Characteristic words used to describe stouts were for example *blackness, oat* or *noir* while reviews of IPAs were classified based on words like *grapefruit*. However, we many of the predictive words were explicit beer names, which might have simplified this task.

8. CONCLUSION

We tried a number of different models for predicting beer and wine ratings from review text. Of these, the two most interesting and successful were linear regression and naive Bayes.

The naive Bayes model was not hugely successful at predicting ratings from reviews. It was able to outperform the most basic naive classifier (always predict the most common label) largely thanks to its prior distribution. But its posterior distribution seems to be too noisy to make much more accurate predictions. Manual examination of the posterior distributions revealed that many of the most probable words for each class were unlikely to have much predictive value (i.e. words like "the", "and", and "with"). Any words that did have predictive value were likely drowned out by the noise of these very common words. We attempted to remove such words from the vocabulary by ignoring words that appeared with similar frequencies across each class. However, our method was not able to completely eliminate such words and only gave small improvements in accuracy. It would be interesting to compare our method to other methods for downweighting common words, such as replacing word counts with the TFIDF [7]. Other changes to the model gave even smaller improvements in accuracy.

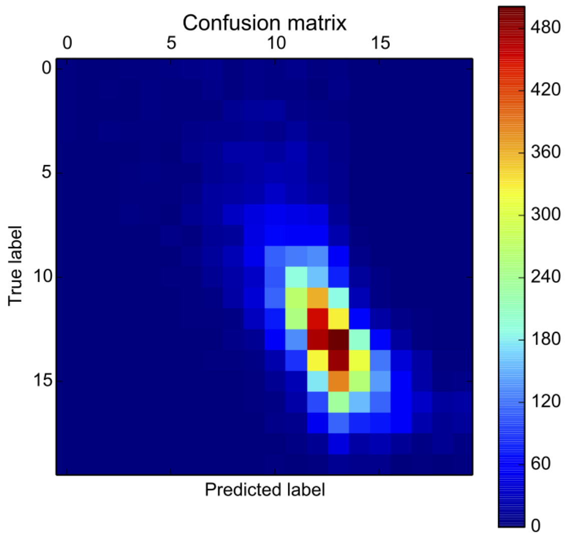

Figure 21: Confusion matrix of RateBeer ratings classified using regression

The linear regression model also did not provide very accurate predictions. Training the model was very efficient, easily scaling up to the full dataset of more than a million reviews on our laptops. However, the model is too simple to capture the complex structure of the text when using unigrams or bigrams as features. Natural language processing, in particular stemming, improved the model only to a minor extent. To gain significant improvements, a richer model like Latent Dirichlet Allocation model might have been better. But even with a linear regression, we were able to distinguish positive and negative sentiment words and get within reasonable accuracy of predictions.

9. REFERENCES

[1] Cellartracker wine rating system. `http://www.cellartracker.com/content.asp?iContent=34`. Accessed: 2015-02-20.

[2] Spacy api. `https://honnibal.github.io/spaCy/api.html`. Accessed: 2015-02-22.

[3] Spacy project website. `https://honnibal.github.io/spaCy/`. Accessed: 2015-02-20.

[4] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[5] G. Ganu, N. Elhadad, and A. Marian. Beyond the stars: Improving rating predictions using review text content. In *WebDB*, volume 9, pages 1–6, 2009.

[6] M. Joshi, D. Das, K. Gimpel, and N. A. Smith. Movie reviews and revenues: An experiment in text regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 293–296. Association for Computational Linguistics, 2010.

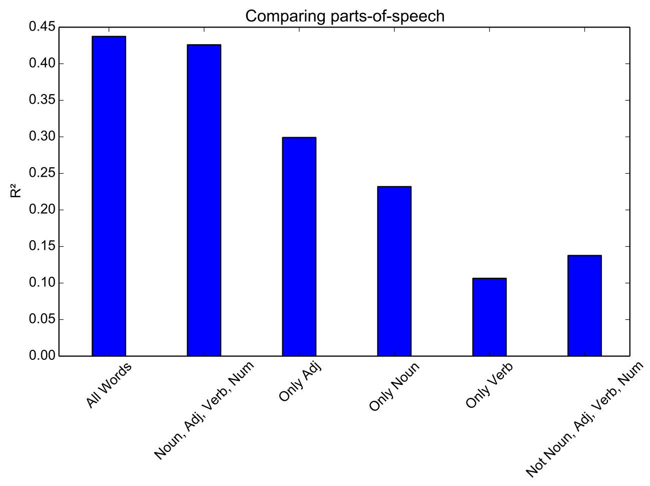

Figure 22: Regression using different parts of speech

[7] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In *AI 2004: Advances in Artificial Intelligence*, pages 488–499. Springer, 2005.

[8] J. McAuley, J. Leskovec, and D. Jurafsky. Learning attitudes and attributes from multi-aspect reviews. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1020–1025. IEEE, 2012.

[9] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908. International World Wide Web Conferences Steering Committee, 2013.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[12] L. Qu, G. Ifrim, and G. Weikum. The bag-of-opinions method for review rating prediction from sparse text patterns. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 913–921. Association for Computational Linguistics, 2010.