HapBeer: A Beer Recommendation Engine
CS 229 Fall 2013 Final Project

Bryan Offutt

December 10, 2013

1 Abstract

This project looks to apply machine learning techniques in the area of beer recommendation and style prediction. The first portion of the project deals with recommendations. With the recent boom in the popularity of craft beers, it has become increasingly difficult to navigate a liquor or grocery store in order to discover new beers that one might enjoy out of the hundreds of options. With so many gimmicky names and brightly colored labels, it has become tough to discern the good from the bad, often resulting in wasted money and a dissatisfied drinker. This project looks to ease this process by providing users with personalized beer recommendations based off of their personal preferences.

The second portion of this project is aimed at helping home brewers rather than general consumers. When making your own brews, people often ask you “what kind of beer is it?”, a question which can be difficult to answer. The style prediction portion of the project classifies a beer into one of thirteen categories based off of the beers alcohol by volume, International Bitterness Units (**IBUs)** and ingredients with the hopes of giving home brewers a quick and easy way to automatically classify their creations.

2 Recommendation Engine

2.1 Data Collection

The data used in this project came from an open source online beer database called BreweryDB [1]. BreweryDB provides a simple API that allows one to obtain a wide variety information about a large number of beers. This information includes things such as International Bitterness Units, alcohol by volume, the brewery the beer comes from, photos of the beers label, and much more. The database includes information for 15,664 beers, but only the **5,868** beers that included IBU information were used in the recommendation portion of this project.

Once the beer information was obtained, the next step was to transform this information in order to create a feature vector for each beer. The features for a beer were based off of four pieces of information: Alcohol By Volume, International Bitterness Units, Category, and Style. Each style belongs to a particular category and contains a subset of the beers in that category. For example, the category North American Origin Ales contains styles such as American Style India-Pale Ale and American Style Strong Pale Ale. While including both category and style adds some redundancy to the feature vector, this redundancy was seen as justified because style is usually a very important factor in a persons beer preference. These four features were represented by a binary feature vector of length 189, where each of the 160 styles and 13 categories were given a binary feature, and IBUs and Alcohol by volume were discretized into features based on ranges ($0<IBU<=10$, $10<IBU<=20$, etc.).

User data was obtained via a simple online survey that asked participants to list beers that they did and did not like. The survey received **72 responses**. However, of these 72 responses only **22** of them listed enough beers to comprise what was determined to be a sufficiently large dataset (13 or more beers). This subset of 22 users was used in the training and testing of the following algorithms.

2.2 Likes/Dislikes Classifier

The first step in the beer recommendation process was to create a classifier that could predict
whether a given user would or would not like a given beer based off of the users past likes and dislikes. In order to solve this task I implemented both a Naïve Bayes Classifier and a Support Vector Machine (**SVM).** Error for each user was calculated by running Leave One Out Cross Validation on each model separately. Once cross validation had been run on every user in the system, the average of the individual user errors was calculated in order to give an overall classification error for each model. By looking at the average classification error per user rather than the overall classification error we hope to create a system that performs well for all users.

2.2 Naïve Bayes Results

My first attempt at a likes/dislikes classifier utilized the

Naïve Bayes algorithm using the multi-variate Bernoulli event model. This initial implementation followed the traditional Naïve Bayes paradigm of calculating the following two values:

$$P(Likes \mid Beer) \;\alpha\; P(Beer \mid Likes)\, P(Likes)$$
$$P(Dislikes \mid Beer) \;\alpha\; P(Beer \mid Dislikes)\, P(Dislikes)$$

Classifications were then made based off of which of these two probabilities was larger. This particular implementation also utilized Laplace smoothing with a smoothing value of .001. Naïve Bayes is known to work surprisingly well for small data sets, and since we were working with limited user data, it seemed a natural fit.

While this model worked relatively well (see Figure 1), I swiftly realized that this was not an ideal system to solve the problem of beer recommendation. When we think about the problem of recommending something, we realize that we are not really worried about classification error, but rather what can be thought of as ***recommendation error.***

$$RECOMMENDATION\ ERROR = \frac{FALSE\ POSITIVES}{TOTAL\ CLASSIFICATIONS}$$

Recommendation error gives us the probability that we would recommend someone a beer that they would dislike. Limiting this number is really the goal of recommendation, even if it means potentially *not* recommending someone a beer that they may have actually liked. The logic behind this is that if we *do not* recommend a beer that a the user would have liked, they will never know the difference. Conversely, recommending a user something they end up hating may leave them saying "this recommendation thing doesn't work right!".

With this in mind it became clear that I needed air on the side of caution and push the classifier towards making negative ("dislike") classifications. In order to accomplish this, I added a ***bias term*** to my estimation of *P* (Likes | Beer). The classification logic now looked like this:

```
if ( $P(Beer \mid Likes)\, P(Likes)$ (Bias) <
        $P(Beer \mid Dislikes)\, P(Dislikes)$ )  {
        classification = dislikes
}
else {
        classification = likes
}
```
Where $0 < biasTerm \le 1$

Bias Value	**Average User Classification Error**	**Recom. Error**
1	.2340	.0804
.5	.2437	.0804
.1	.2563	.0714
.01	.3743	.0595
.001	.3212	.0387
.0001	.3504	.0238
.00001	.3650	.0238
.000001	.4196	.0149
.0000001	.4704	.0089

Figure 1: *Classification and Recommendation Error (of LOOCV) for Assorted Bias Terms*

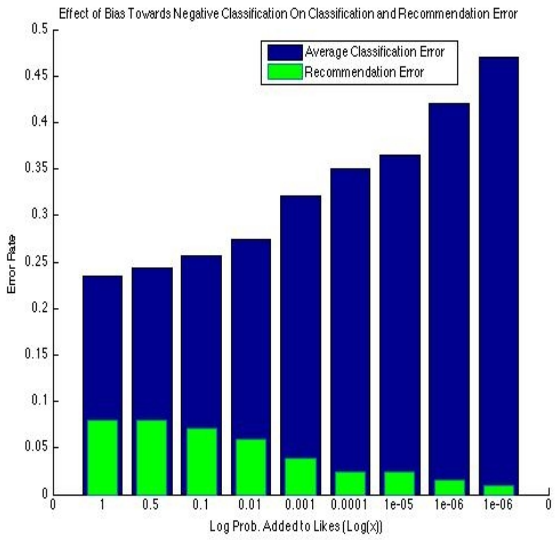

Figure 2

While the addition of this bias term raised classification error and false negatives (as expected, see Figures 1, 2), it considerably lowered the classifiers performance in terms of recommendation error. Furthermore, by picking a bias term of around .001, you can obtain the advantages of this great recommendation error while still maintaining solid classification performance.

2.2B Support Vector Machine

The SVM portion of this project was implemented using a combination of the LIBLINEAR [2] and LIBSVM [3] MATLAB libraries. After trying a variety of kernels, I found that a linear kernel was the most effective on the training data. Since we have a large number of features and a very small number of training examples, it makes sense that we would not want to map our features into an even higher dimensional space.

	NB (No Bias)	SVM (Lin. Kernel)
Average User Class. Error	.2340	.2110
Recom. Error	.0804	.0804

Figure 3: *SVM and NB Errors*

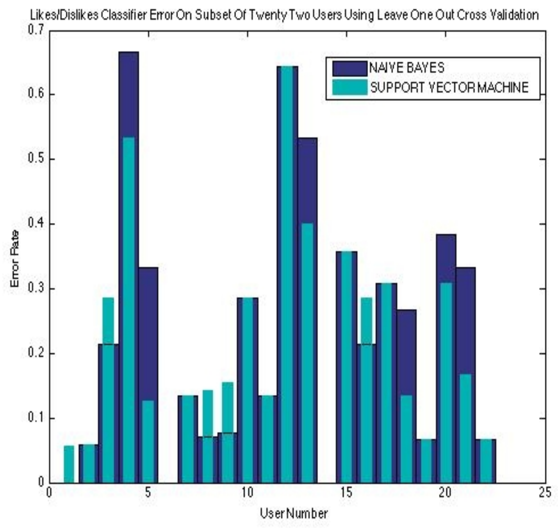

Figure 4: *Graph of SVM vs NB Errors*

From the results we see that the SVM does a slightly better job properly classifying a beer as a "like" or "dislike" than the Naïve Bayes classifier does. Given the simplistic nature of Naïve Bayes, this is not a very surprising result. However, it is worth noting that with the addition of the bias term Naïve Bayes is able to significantly outperform the SVM in terms of recommendation error.

1.3 Final Recommendation

Once the the Likes/Dislikes Classifier had been trained and tuned on the training data, it became time to actually make a recommendation. This process followed the following steps:

1. Run K-means using all of the beers in the database in order to identify clusters of beers that have similar features. (Ten centroids were used, resulting in 10 distinct beer clusters.)
2. Create a vector that is the sum of all of the feature vectors corresponding to beers the user likes.
3. Divide the resulting vector by the total number of beers the user identified as a "like", creating that users "average" beer, or a ***user centroid.***
4. Identify which of the 10 k-means centroids this user centroid was closest to.
5. Choose a random beer from the list of beers that are assigned to the centroid you found in part 4.
6. Run the Likes/Dislikes classifier on this beer. If it is determined the user will like this beer, recommend it. Otherwise repeat step 5.

By first using k-means to cluster beers and then recommending beers from the cluster which our user centroid is assigned to, we assure that the beers we run through our Likes/Dislikes classifier are similar to the users past preferences. This saves us the time of running our classifier on beers that are very different from the users preferences (and thus likely to receive a negative classification), while simultaneously providing our recommendation with an additional layer of assurance.

1.4 Discussion

Overall, I think the recommendation engine performed fairly well. The cross validation error that we obtained for the Likes/Dislikes engine was surprisingly low, even when using a relatively simple algorithm such as Naïve Bayes. In addition, after running the full recommendation system for a number of friends, it seems that this too performs fairly well. However, this is tough to test quantitatively, since the odds of the person having had the recommended beer is relatively small.

With that said, this recommendation system is far from perfect. While the average classification error and recommendation error was relatively low, there were still a few users on whom the system performed rather poorly. After examining the data, I determined that the system performs much better (as you'd expect) on people who's likes and dislikes have very distinct features. Users who's likes contained a variety of different beer types tended to do much worse in terms of classification error. Another very common issue was that a lot of users had two beers that were *incredibly* similar feature wise but opposites in terms of classification. For example, some users would list Bud Light as a like and Coors Light as a dislike when, from an objective perspective, these two beers are almost identical. Having two nearly identical beers with opposite training labels can make things very difficult on the learning algorithm.

Furthermore, because the SVM performed better on classification and the Naïve Bayes classifier with a bias term performed better on recommendation, we are faced with the question “which one is better?”. In the long run it really depends on how adventurous you are: take the safe Naïve Bayes route and potentially miss out on great beers, or take the SVM route and potentially get some beers you don't like. While they both have their advantageous, I decided that Naïve Bayes was the better option in this situation because it makes it incredibly easy to obtain high quality *recommendations,* which is the end goal of this project.

3 Style Prediction

3.1 Data Collection

Similar to the Recommendation portion of this project, the style prediction portion also utilized data from BreweryDB [1]. However, this portion used only the **794** beers that had ingredient information in the database. The information about these beers was broken up into a binary feature vector based on Alcohol By Volume, IBUs, and Ingredients. There are a number of ingredients listed in the database that are not actually in any of the beers, and these ingredients were left out in order to decrease the size of the feature vector. Because these ingredients are not in any of the 794 beers in the training data, these features would always be zero, and thus would not add any real value to our system.

In order to make the features in our vector binary, the alcohol by volume and IBU features were discretized by range just like in the Recommendation feature vector. Additionally, each ingredient type (Admiral Hops, Amarillo Hops, etc.) was given it's own binary feature which was 1 if that ingredient was present and 0 otherwise. The resulting vector had a total of 701 features. 684 of these features were ingredients, while the remaining 17 were the discretized features for ABV and IBUs.

3.2 Method

In order to solve the problem of style prediction, I utilized a multi-class Naïve Bayes classifier. This classifier worked in the same way as the one used in the Likes/Dislikes Classifier, except it chose from 13 different categories (Figure 5) rather than simply 2 (like or dislike). The category with the highest likelihood as determined by the classifier was selected as the prediction.

British Origin Ales	**Irish Origin Ales**
North American Origin Ales	**German Origin Ales**
Belgian And French Origin Ales	**International Ale Styles**
European-Germanic Lager	**North American Lager**
Other Lager	**International Stylrs**
Hybrid/Mixed	**Mead, Cider, and Perry**
Other Origin	

Figure 5: *Possible Styles*

To control for unseen features, Laplace add one smoothing was initially implemented. This initial algorithm performed very poorly, and I swiftly realized it was due to this smoothing value. Because each category has only a small number of training examples with it's respective labeling, the number of beers in this category that have a given feature is fairly low for the majority of features. In addition, due to the size of the feature vector, a large number of features are ***never*** seen in a beer with this labeling. In this case add one smoothing places too much value on these never before seen features, giving them almost as much weight as features that *have* been seen, though only a small number of times. Thus it became apparent that a smaller smoothing value was in order. In order to solve this I ran the algorithm using a number of different smoothing values and chose the best one using 70-30 cross validation. (Figure 6)

3.3 Results

Considering the large number of categories from which the algorithm had to choose from and the simplistic nature of Naïve Bayes, it is not surprising that the classifier struggled. On the held out validation set, the classifier was able to correctly guess the category **in one guess 51.05%** of the time, **in two guesses 34.31%** of the time and in **three guesses 25.52%** of the time (Figure 6). The drastic increase in accuracy with each additional guess is likely due in part to the fact that 46% of the training examples were North American Ales. Thus, the prior probability of North American Ale was very high while the prior of the rest of the categories was relatively low. As a result, North American Ale was very frequently (and often incorrectly) the classifiers first guess, while the correct category was the second or third guess. Another potential source of error is the fact that some groups of categories, such as North American Origin Ales and Irish Origin Ales, have relatively similar characteristics, which can make it difficult for the learning algorithm to distinguish between the two.

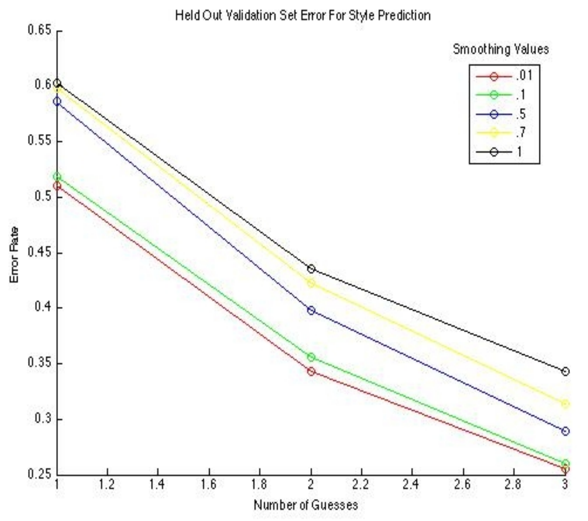

Figure 6: Style prediction held out error using a variety of smoothing values.

4 Conclusions and Future Work

While the beer recommendation and style prediction engines described in this paper are a good start, there is still much that could be done to improve them. The first and most obvious improvement would be to obtain more training data, particularly in the user department. While it was great to have 72 people respond to the survey, it was a real shame that so few of them included enough beers to comprise a decent data set. 22 users is simply not enough to be sure that we have a recommendation system that is reliable. Having a larger number of users would be incredibly helpful in assuring the system performs as well as possible on a variety of different preference types.

In addition to simply obtaining *more* user data, it would be great to have a more *varied* group of users. The majority of respondents to my survey were 21-25 year old white males, which is not representative of society as a whole. Data on users of different genders, ages, and ethnic backgrounds would help paint a more holistic picture.

Another great way to improve the recommendation system would be to add information about the users themselves into the recommendation. One piece of information that I believe would be especially useful is the users geographic location. Looking through the data (and considering my own preferences) it became apparent that people have a lot of hometown pride when it comes to beers. The quality of your home city's breweries is a point of pride for a lot of beer drinkers, and their likes often reflect this. Finally, there is one key feature I would add to our feature vector: PRICE. Price seemed to be a ***huge*** factor in peoples likes and dislikes, with cheaper beers tending to end up in the latter group. This feature could be a small but powerful addition to our recommendation system.

In addition to recommendation improvements, there are also a number of improvements that could be made to Style Prediction. While my Naïve Bayes system was able to at least give a general idea of what category a beer would be in, the results were certainly not up to what you would want in a deployable system. Trying a new, more complex model such as Soft Max Regression may very well yield better results. In addition, obtaining a larger training data set with more diverse training labels would be a great help. As I said earlier, the majority of the beers (with ingredients) from BreweryDB were North American Ale's. This is likely a result of the fact that Brewery DB is an American based website, and thus favors American craft brews, many of which are some sort of Ale. It would be nice to have a bit more variety in our data.

5 Acknowledgements

[1] *Brewery DB*. 10/2013-12/2013. Web. http://www.brewerydb.com/

[2] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. *LIBLINEAR: A Library for Large Linear Classification*, Journal of Machine Learning Research 9(2008), 1871-1874. Software available at http://www.csie.ntu.edu.tw/~cjlin/liblinear

[3] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines.* ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm