CSCE 5543: Homework 2

April Walker
adw027@uark.edu

University of Arkansas,
Fayetteville, AR, 72701, USA

Disambiguation in Amazon Review Corpus

0.1 Introduction

This assignment explores disambiguation. In our case, we use pseudo-words to differentiate between two sometimes more and sometimes less related words. Our corpus gives us rather varied sample sizes for each word, adding some interesting bias when classifiers are compared. While the focus of this project is the Naive Bayes Classifier, I also explored classification using a Gradient Boosted Machine and a Neural Network (the latter more as a joke but still it worked remarkably well).

0.2 Methodology and Results

For this assignment I kept preprocessing rather minimal, in fact, I copied over the code from the last assignment to clean the corpus. I chose to keep the stop-words as I felt they wouldn't very much hurt the Bayesian Classifier (that is a conditional probability neutral word will have little sway one way or the other) while taking them out may give an unrealistic view of the surrounding context. While not explored, I would expect a balanced set with a smaller window but stop-words removed to perform optimally for the Bayesian Classifier.

For the Neural Network (NN) and Gradient Boosted Machine (GBM), I fed in the data as 20 individual predictor columns. Because of this, I feel it might benefit more from an expanded window, as they tend to pick up on feature value on their own.

In the early stages of this project, I had planned on writing the entire project in `python` thus my training and testing dataset preparation more mirrors a standard approach but seems incredibly unnecessary for the Naive Bayes Classifier. I went ahead and left this "as is". This bit included combining training and testing sets from each of the words in a word-pair then shuffling them. Of course, since the Naive Bayes simply takes conditional probabilities of words in the training set by counting word frequencies, this doesn't impact the result.

Another personal choice was to use my previously written function `get_ngrams()` to develop hashtables with word counts for the Naive Bayes Classifier even though it's a bit nondescript of the action. I felt this is more in the spirit of software development. In a more corporate setting, you'll likely (hopefully?) make extensive use of the company's codebase and your own. It may be less copy-pasting and more calling their source code but the heart of this choice is the same.

While the mathematics of the Bayesian Classifier has been covered in class, I will briefly review my main classification algorithm. Below is the equation for the approximate conditional probability of a word W given it is in the context of some classification C. Here $|V|$ is the total size of the vocabulary of context words from the training sets of both classifications, $count(C)$ is the total number of tokens in the training set for the classification C and $count(w, C)$ is the number of

times word w occurs in the training set for classification C.

$$\hat{P}(w|C) = \frac{count(w, C) + 1}{count(c) + |V|}$$

From this, we can calculate what we truly want to know - the probability of a classification given a set of words. The equation for this is below.

$$P(C|W) = \prod_{i=1}^{|W|} P(w_i|C)$$

Where W refers to the entire set of context words for an observation we wish to classify and w_i refers to some specific word.

To test my algorithm, I used the context words given in class when discussing the distinction between "bass" the fish and "bass" the instrument. After this test passed I was ready to test my methodology on the full set. My results from this are more thoroughly shown in the appendix.

The Bayesian Classifier performed rather uninterestingly. While the accuracy was always above 50%, its best performing cases measured strictly my accuracy further point out its tendency to simply pick the category with the highest probability. This is obviously clear with "car-bike" where it performed at 95.9% accuracy. It failed to ever recognize and occurrence of "car" but still miss-classified some cases of "bike".Its second-best performing case "kitchen-cough" with an accuracy of 68.2% similarly failed to recognize any cases of "kitchen" but miss-classified "cough" 5 times. All others performed remarkably close to 50%. What I found even more surprising is how irrelevant changing the window for my context words was. While they are not listed in appendix I found that my accuracy for a surrounding context of 15 instead of 10 gave me 52.1% accuracy for "night-seat", 65.0% accuracy for "kitchen-cough", 94.3% accuracy for "car-bike", 50.0% accuracy for "manufacturer-bike", 50.3% accuracy for "big-small" and 49.5% accuracy for "huge-heavy". On the other side of things a surrounding of 5 instead of 10 gave me 51.9% accuracy for "night-seat", 70.0% accuracy for "kitchen-cough", 94.4% accuracy for "car-bike", 53.7% accuracy for "manufacturer-bike", 50.7% accuracy for "big-small" and 50.5% accuracy for "huge-heavy".

0.3 Methodology and Results Continued - Neural Network and Gradient Boosted Machine

I decided to split this section up as this portion of the assignment was unnecessary. While the NN was chosen due to its discussion in class, the GBM was chosen more out of pure favoritism (the worst reason to pick a lassifier,, but a reason nonetheless). As previously stated, the words were read in as 10 columns of variables. While this is a potentially weird approach to variable creation, I'd expect position itself to give some context clues related to word-pairs. While not explored, additional methods of variable creation from these context words might prove even more fruitful than the results for this project.

For both the NN and GBM I chose a rather minimal grid search without cross-validation. Since a GBM is pretty much a random forest boosted by gradient descent, its hyper-parameters should be thought up pretty much in these terms. For my GBM I allowed the `learning_rate` to vary as 0.01, 0.05, or 0.1 and the `max_depth` to vary as 3, 10, or 30. I chose `learn_rate_annealing` to be 0.99. The `ntrees` was kept at 100 to be decently through but also speedy. Between the annealing and chosen `learning_rate` values, you can already expect these values to overfit. This was partially on purpose so I could abuse the "accuracy" metric on unbalanced words. I was actually rather surprised by how well the GBM performed on the more balanced sets. While the training accuracy is not shown in the appendix, it was generally.... very overfit. The accuracy rate was almost always in the upper 90%'s and rather often up to 100%. This was, of course, more exploratory than anything else.

The neural network was almost explicitly a joke but still performed remarkably well. Considering my definitely-not-deep-learning-sized dataset I bumped the hidden layers down from the [200, 200] to alter between [1], [2], and [5]. The `epochs` hyper-parameter was allowed to be either 3, 5, or 10. My NN tended to perform rather close to my GBM while taking a fraction of the time to train. Maybe my `ntrees` was still a little overboard on top of my absurd `learning_rate`...

0.4 Final Thoughts

Overall my Naive Bayes Classifier was less than impressive. My initial thought that stop-words wouldn't be a killer was likely inaccurate (I'd bet somewhat due to the unbalanced datasets... but probably just in general). Messing around with the window of context made inconsistent and minimal impacts on my Naive Bayes Classifier.
On the other hand, even with the minimal preprocessing and somewhat unformatted method of variable creation, my machine learning based classifiers performed rather decently. In the future, spending more time exploring various methods of preprocessing and dataset balancing and spending less time messing around with fun machine learning classifiers might provide a more robust disambiguation classifier.

0.5 Appendix

	Naive Bayes		NN		GBM	
	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Night	73/107	68.2%	105/107	98.1%	106/107	99.1%
Seat	11/43	25.6%	31/43	72.1%	27/43	62.8%
Nightseat	84/150	56.0%	136/150	90.1%	133/150	88.6%

	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Kitchen	0/2	0.0%	0/2	0.0%	0/2	0.0%
Cough	15/20	75.0%	20/20	100.0%	20/20	100.0%
Kitc...ough	15/22	68.2%	20/22	90.9%	20/22	90.9%

	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Car	0/20	0.0%	0/20	0.0%	3/20	15.0%
Bike	635/642	98.9%	642/642	100.0%	642/642	100%
Carbike	635/662	95.9%	642/662	97.0%	647/662	97.7%

	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Manufacturer	10/20	50.0%	11/20	55.0%	12/20	60.0%
Bike	10/20	50.0%	19/20	95.0%	18/20	90.0%
Manu...bike	20/40	50.0%	30/40	75.0%	30/40	75.0%

	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Big	164/281	58.4%	159/281	56.6%	155/281	55.2%
Small	100/237	42.2%	195/237	82.3%	215/237	90.7%
Bigsmall	264/518	51.0%	354/518	68.3%	370/518	71.4%

	Success Rate	**Accuracy**	**Success Rate**	**Accuracy**	**Success Rate**	**Accuracy**
Huge	38/98	38.8%	80/98	81.6%	80/98	81.6%
Heavy	76/125	60.8%	110/125	88.0%	107/125	85.6%
Hugeheavy	114/223	51.1%	190/223	85.2%	187/223	83.9%