Data Analysis and Artificial Generation of Amazon.com Product Reviews

John Feilmeier

5.1: Source files

Source files, both in CSV and ARFF format, are available at my project's Github repository[1], in the **arffFiles** and **parseResultCSVFiles** directories. The files used are **Big_Bigram_goodbad.arff** and **Big_Bigram_goodbad_SELECTED.arff**[1] for the bigram data, and **Big_Smart_goodbad.arff** and **Big_Smart_goodbad_SELECTED.arff**[3] for the 'smart' data.

5.2 Initial Documentation

5.2a: Additional Data Collection

After some analysis, I found that my initial data collection was unbalanced. I had collected the first 50 pages of reviews, sorted by those most recently written. As it turns out, most of the oft-reviewed products on Amazon.com that I found have an average rating of over 3.5 stars, with the bulk of those reviews being 4 - or 5 stars. I tried removing some of the positive reviews from my dataset to make the data more egalitarian, but the resulting dataset was small enough that the quality of predictions began to fall.

Returning to the reviews page for some of my chosen products on Amazon.com, I found that I could sort the reviews by 'Positive' (4 or five stars) or 'Critical' (less than 4 stars). I went back to my source URLs and had them sorted by those that had highest rating ('Top Rated'), and separately scraped the first 50 pages of reviews for both Positive and Critical reviews, leaving the computer to chug through the scripts while I got myself a snack. When I returned, I was horrified to find that not only were most of the entries for product reviews totally empty, but that I had hard-coded the output files for the data and effectively wiped out all of my data! Digging deeper, I found that the titles of these missing product reviews were all 'Robot Check'. I had removed the 'sleep' instruction in my web-scraping script (to save time) and Amazon's bot checking algorithms caught and tagged my computer and connection string as a web spider!

Fortunately, I had not pushed these latest changes to my GitHub repository and could retrieve my previous data. Unfortunately, I still needed more reviews. I searched around the internet and found that I was not the first to make this mistake, and that by changing my connection string (and reintroducing the sleep timer) I might be able to start scraping again. With a 3 second rest between website requests and a new name, I was able to successfully retrieve the increased dataset that I used for the rest of the project, with only 17% more positive reviews than negative, compared to the nearly 75% ratio of the original data.

In my initial tests of the newer data scraping script, I collected a smaller sample dataset consisting of positive and critical reviews for 3 different coffee makers. I wanted to see what the difference in the active bigrams would be when all the product reviews were for a similar product. This was also to save myself the hours it would take to collect bad data in the event that my new user string did not work. This run did work, and the active word and word pairings reflect the source products, with 'coffee', 'machine', 'water', 'the coffee', 'coffee maker', 'this machine', 'of coffee', and 'mr coffee', among the highest occuring words and bigrams in the reviews.

5.2b: Goals Reached

I am pleased with the accomplishments made in this project. While I did not achieve above 80 percent for any of the classification or prediction methods, I have managed to collect, process, create and apply models, and generate novel reviews. I tried to vary my approaches to these different goals, processing my data into two main camps: a simpler 'bag-of-words' set of data, and a 'smarter' set that utilizes different natural language processing techniques to create a dataset *about* the data, instead of counting words.

FIND CONSISTENCIES OF REVIEW TO RATING

I am happy with the results of this aspect of my project. I didn't think I would get over 60% accuracy given the dataset I am working with. People all over the country, at different ages, different literacy levels, different languages and accents, with different opinions, all writing and rating these products they have bought. In spite of all that, there are things that we have in common in the way we express ourselves on Amazon.com. Of the two approaches I took, the 'bag-of-words' had better accuracy. This is probably due in part to my beginner knowledge of NLP, and therefore the failings of my 'smart' dataset. It may also be related to the nature of these online reviews, where grammatical analysis is more evenly spread across all products and ratings, and relate more to the individual reviewer than the rating or product itself.

FIND CONSISTENCIES OF REVIEW TO PRODUCT

This goal was abandoned in favor of spending more time on the text generation. The initial attempts to classify reviews by their product were rarely better than a random guess, and the removal of the product name dropped the accuracy of classifications by less than one percent. This may have been due to the random list of products I chose, or the number. I'm sure that if I had scraped data of only two products that the results would have been better, but it seems like the choice of the product would matter more so than the differences between the products.

GENERATE REVIEWS

> “i'm getting ready however it does not dissappoint full of the chicken and he loved it however with other hair supplements on the bottom of the tabletop of the quality is great product at a great product can't beat the price they are quite deep it has started togrow my i”

-Markov Chain text generator

> "ld it was a gift for my daughter for a good product i had to return it and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my “

-LSTM Neural Net and over 20 hours of my life

This aspect of the project was fascinating and produced some very interesting results. I used both LSTM Neural net and a Markov Chain technique to produce text. Building the model for the LSTM took 18 hours to complete! The Markov, on the other hand, was very quick and produced more legible results.

LEARN ABOUT NLP

Not only did I catch on to some of the basics of Natural Language Processing, but I also learned to use some new tools in Python and WEKA. My work on this project has confirmed my suspicions that NLP is an incredibly interesting field of study, and I hope to have the opportunity to learn more about this realm of computer science.

5.2c: Machine Learning/ Modelling Techniques

On the Bigram version of the ‘bag-of-words’ dataset, I used the WEKA ‘Bayes’ functions NaiveBayes, BayesNet, and NaiveBayesMultinomial, the WEKA ‘functions’ Logistic and MultilayerPerceptron, the WEKA ‘rules’ OneR and Decision Table, and the WEKA ‘tree’ functions RandomTree and J48. Most of the results came in near the 70% mark of accuracy. The ‘rules’ and ‘trees’ families of functions took some time to evaluate the data after building a model in the full, 200-attribute dataset[1].

METHOD	% CORRECT ON FULL BIGRAM DATA	% CORRECT ON SELECTED ATTRIBUTES IN BIGRAM DATA	% CORRECT ON FULL 'SMART' DATA	% CORRECT ON SELECTED ATTRIBUTES IN 'SMART' DATA
NaiveBayes	70.3933 %	65.99 %	59.9911 %	60.0452 %
BayesNet	70.3933 %	66.2646 %	62.3125 %	60.7289 %
NaiveBayesMultinomial	73.5314 %	66.1665 %	56.9026 %	59.6911 %
Logistic	72.7175 %	66.3528 %	62.7256 %	61.2551 %
OneR	54.3003 %	54.3003 %	59.1206 %	59.1206 %
DecisionTable	63.9011 %	64.5092 %	62.5486 %	60.5075 %
RandomTree	67.4512 %	65.8037 %	60.1682 %	59.4206 %
J48	71.5505 %	65.8527 %	63.6158 %	60.9649 %
MultilayerPerceptron	60.1549 %	64.2542 %	63.3601 %	61.7961 %

Fig 1: Accuracy results of various WEKA algorithms on both sets of data

I used the 'Select attributes' tab[2] in WEKA Explorer to pick those attributes the algorithm finds most useful, here using the default BestFirst method. It chose 23 attributes:

('i', 'love')
('love', 'this')
('in', 'the')
('at', 'all')
('did', 'not')
('does', 'not')
('love', 'it')
('would', 'not')
('the', 'box')
('great', None)
('do', 'not')
('the', 'product')
('the', 'first')
('for', 'my')
('great', 'product')

('will', 'not')
('to', 'return')
('easy', 'to')
('i', 'thought')
('works', 'great')
('of', 'the')
('the', 'same')
('not', 'work')

A few of the entries, such as “in the” don’t immediately make sense, but I was struck how many of the reviews contained this similar language. Removing all other attributes than these and the target, I ran the same algorithms again with this smaller dataset(fig 1). The results were fairly close for some methods, and the DecisionTable actually got more accurate, most likely due to decreasing the number of attributes nearly 80%! The other methods lost some accuracy due to a drop in granularity in the data, but the resulting trees and results became much more readable. Whereas the size of trees from the full dataset were 500 and 8000 (!) for J48 and RandomTree, respectively, those numbers came down to 49 and 1035, a huge drop compared to the slight decrease in accuracy.
The size of the full data put some strain on my computer, with the MultilayerPerceptron taking 23 minutes to build a model and longer to calculate the folds for classification. This was somewhat ameliorated by decreasing the number of attributes, but the RandomTree still remained large and nearly incomprehensible. I think this is due to the nature of the data; these words or bigrams don’t necessarily relate to each other in any way that is reasonable to the computer, just as hearing people speak a foreign language doesn’t mean one comprehends it.
The ‘NaiveBayesMultinomial’ function provided the most accurate results for the ‘bag-of-words’ dataset. It is a specific implementation of the NaiveBayes algorithm that assumes the values being encountered are a count, such as a word count, which is exactly what they are! I thought the NaiveBayes algorithm would perform best, since the attributes are very independent of one another. It makes sense that the best performer would be a naive Bayes made specifically for counts.

For the ‘smart’ dataset, I used the same functions as the ‘bag-of-words’ data, as well as the ‘meta’ function Bagging, which allows the user to utilize one of the other functions over and over again, averaging the different models produced to minimize variance between different possible outcomes of an unstable algorithm, such as J48 or RandomTree. These results did not improve very much upon the straight J48 and Random Tree results.
Where the other dataset had extra attributes, this one could probably benefit from some more. This data is an attempt to comprehend the language used by the reviewers, instead of regurgitating them all into a Magical Prediction Box that turns them into results.

5.2d: New Techniques Used

MULTILAYER PERCEPTRON

In WEKA, I used MultilayerPerceptron for the ‘bag-of-words’ datasets. I thought that since I had maxed out the acceptable number of attributes with nearly random bits of information, a simple back-propagated neural network might have good results. It fared O.K., but the run for the full dataset took a few hours ***???***. It actually came in at a close second for the ‘smart’ dataset, narrowly losing to the J48 tree.

LSTM NEURAL NETWORK

I was interested in using a neural net after seeing quirky and interesting applications of these methods for generating valentine’s heart messages and photographic images. I followed Jason Brownlee’s tutorial[4] on his machine learning blog, using the first 200,000 characters in the total compiled text of the critical reviews from my raw dataset. Training the model with 50 epochs was to take over 13 hours, so I left my computer to run while my wife and I ran errands. To my horror I found that my computer had restarted and killed the process! When I did finally get the program to run, it took around 10 hours. While the results were not the enlightened prose I had hoped for, it is fascinating to see what these techniques can accomplish by building a sentence character by character. From the seed text:

> "uality problem here that the manufacturer is not addressing instead in response to so many of the re"

We get the result:

> “cord player is sounds great and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the sound quality is a gigt for my daughter and the soun”

Which is, granted, pretty repetitive after 5 words. The cool thing is how the network completed the seed of ‘re’ to generate the word ‘record’ and began to create sensical words that talk about a record player. It then reverts to the “and the sound quality is a gigt for my daughter” mantra. I would like to experiment with this technique more, although perhaps not with text generation as the time required to build a model somewhat stymies the creative spirit of experimentation. Utilizing Amazon Web Services or some other remote processing service might make the experience more painless.

MARKOV CHAIN

Comparing the LSTM with a Markov Chain, Markov Chain is a much more effective tool for generating text! It is much simpler to implement and the time required to run an implementation of it in Python is almost trivial. Also, the results are fantastic!

> “them for all of the crystal enthusiast or healer a must have if you don't have plastic components so that why i thought i thought it was a bargain highly recommend the vitamins are the best blender i thought you could just make sure it takes a lot of the while”
>
> “every vinyl record i've tried in the day i thought it was jammed when it comes to return it and it worked fine and thinning over a year my hair is not more than twice a day over time i thought it would not put it in gallons to return it there are other font types i thought i'd give this jbl a good size people are gonna love it and i thought but well eventually”

-Two examples of Markov Chain generated text

It works be generating a dictionary of words from a text. The keys are a pair of consecutive words, and the values are lists of possible third words based on a model of the text. I followed a tutorial[5] to get the basics, and added an option to prefer those word pairs that show up in the top-selected Bigram attributes from the ‘simple’ dataset I generated. These examples are based on the positive reviews, with the option to favor selected words. Here is a taste of random choices:

> “the bottom of the blades are relly sharp be careful and take down there are no other explanation for spiritual info on cleansing and storage bins now have a big pot would not recommend getting this purse i thought fits great product also works great product my co worker to return it and was packed very well and just wanted them to return it there these probably fit or water bottles inside if that is little”

Since whole words are chosen based on previous use, even the incoherent results are legible and almost syntactically correct. With some more work on the selected attributes option, and trained on a specific type and rating of product, this could generate useful customer reviews.

5.2e: Commercial Applications

TEXTUAL ANALYSIS

My processing and analysis of the raw review data brought out interesting features. Words and word pairs that correlate to a positive or negative review could be used by almost any reviewed business or product in a customer satisfaction survey, with key words providing insight into possible problem areas or where there is success. It could also allow for monitoring on communications (forthright or clandestine) to gauge sentiment of the parties involved based on the frequency of key words. With the huge volume of recorded textual back-and-forth generated in the modern era, natural language processing and sentiment analysis could become a powerful tool in a company or government's ability to maintain control over those people within its purview.

TEXT GENERATION

I have only dipped my toe into the dark waters of automated content generation, but the results are still compelling. Aside from creating goofy entertainment bots[6], these methods could streamline many aspects of bureaucracy. Forms requiring a written review or rundown of events could be automatically generated, saving hundreds of hours of monotonous work and allowing important employees to spend their valuable time doing more worthwhile tasks. Text generation could be used in the marketing and entertainment industries for small companies without the budget for a dedicated advertiser to help create the next viral marketing campaign, or by print-on-demand t-shirt and mug companies to find a best-selling product. Scripting and web programming could be done automatically based on a few requirements and a supervised and weighted code-generation system. As powerful computing becomes more accessible via cloud computing services[7], the cost associated with performing this kind of science comes down, allowing more developers, both professional and hobbyist, to work on this problem.

5.2f: Other Interesting Items

I found that working from scratch was a valuable experience. Actually retrieving the raw data and getting to a useful state emphasized the importance of preprocessing in data mining, and it was where I did most of the work. I had some ideas about what would be meaningful in a product review, and took those ideas to the 'smart' dataset, which underperformed against the simpler 'bag-of-words' style dataset. It seems like the only barrier to better performance is how well one preprocesses the data.

I used WEKA predominantly in this project, as I am familiar with it and I find the GUI helpful when trying to comprehend what is happening to the data as the algorithms run. Other tools, such as ScikitLearn for Python might be able to take more attributes, as WEKA would not read CSV files

with over 202 attributes. This limited my dataset , but based on the results of removing all but 23 word pairs only losing a few accuracy points, I think that the number of attributes would not make much of a difference.
This project has widened my perspective on data analytics and data mining. The idea that a novice such as myself can pull thousands of ratings down from webpages on Amazon.com, process them and be able to gain insight into what makes a positive or critical review, let alone generate reviews from this collected data that *almost make sense* is incredible. Amazon.com reviews are a fairly obvious and low-hanging fruit, but there are large stores of data all over the internet, both public and private. On one hand this is exciting: for many of the problems in various fields, the answer may be out there in the hidden patterns of commerce, communication, and cohabitation we generate every day on our phones and devices. On the other hand, when a robot can generate realistic product reviews, ad campaigns, or television scripts, how will that change our culture and our media? These changes are happening in my lifetime, and through this project I have found that I want to be a part of this conversation.

References:

[1]: Link to results and full output of full bigram data analysis:
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/bigram_results_200_attrs.txt ,
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/bigram_results_selected_attrs.txt
[2]: Link to result of WEKA's select attributes on bigram and 'smart' data:
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/bigram_selected_result.txt ,
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/smart_selected.txt
[3]: Link to results and full output of 'smart' data analysis:
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/SMART_RESULTS_FULL.txt ,
https://github.com/Jacktavitt/product_review_fun/blob/master/source/wekaREsults/SMART_RESULTS_SELECTED.txt
[4] : Jason Brownlee, Machine Learning Mastery,
https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/
[5]: Shabda, Agiliq,
https://www.agiliq.com/blog/2009/06/generating-pseudo-random-text-with-markov-chains-u/
[6] : Subreddit Simulator, a subreddit populated exclusively by bots
https://www.reddit.com/r/SubredditSimulator/
[7]: https://cloud.google.com/gpu/