# Sentiment Analysis for Amazon Reviews

Thida Aung
Santa Clara University
taung@scu.edu

Sanjay Kaliyur
Santa Clara University
skaliyur@scu.edu

Erik Trewitt
Santa Clara University
etrewitt@scu.edu

**Abstract**

Product reviews are the basis for purchasing trends in the consumer marketplace. With the ever expanding internet, manually looking at reviews is becoming an increasingly tedious and impossible task. Thanks to the certain technologies like Machine Learning, we can streamline the process of determining if a product is good or bad by allowing a computer to determine the sentiment of a review based on a large data set of examples.

**Keywords**

Product reviews, star rating systems, supervised, amazon reviews modeling, logistic regression, naive bayes

## 1. Introduction

Product reviews are key influencers in making shopping decisions online. Rarely do people purchase a product on any website without checking the reviews first to see how others who purchased the same item rated for the product. For this reason, the ability to determining the sentiment of a review easily is incredibly valuable for both consumers and sellers. Machine Learning is the only way to adequately solve the sentiment determination problem because there are enormous number of reviews online.

Our process of using Machine Learning began with determining what kind of Machine Learning algorithm we wanted to use. This part boils down to whether we wanted a supervised or unsupervised learning algorithm. A supervised algorithm is one that takes in a data set that it “trains” itself with. Essentially these are examples of the kinds of data that we want to explore and how they are classified in reality. By doing this, when we pass in the data set that we want to explore, our algorithm will compare the data that we want to know more about with the data that we trained it with. An unsupervised algorithm doesn’t use any training data and uses other techniques to determine the true nature of the data that we want to know more about. For our purposes, we decided that a supervised algorithm was the more useful option after researching few related works which will be discussed after this section.

The next step was determining which supervised Machine Learning algorithm we wanted to use that fits our data best. We decided to go with Logistic Regression, an algorithm that focuses on binary classification of data (e.g. hot/cold, good/bad, etc). We also decided that one algorithm would not be sufficient in determining the true accuracy of our test. So we decided that we would also implement a Naive Bayes' Classifier algorithm. This algorithm is based on Bayes' Theorem, a statistical model that determines the probability of an event based on prior probabilities of events related to the specific event that we are testing. We will go into more detail on these algorithms in the Methodology section.

As far as the type of data is concerned, we decided to explore the sentiment distribution of Amazon Video Game Reviews. This data set contains product reviews for various video game-related products on the Amazon marketplace. A typical Amazon review comes in the form of a star-rating, in this instance a 1-5 rating, and an optional comment. For our testing, we decided that the best way to determine the accuracy of our model was to pass in the review comments and have our model guess the corresponding star rating for each product in the data set. Then we would compare the predicted rating with the actual 1-5 rating left by the original reviewer.

Because we used supervised Machine Learning algorithms, we needed a proper example data set to help train and test our model to estimate the review sentiments. From our video game dataset, we decided to split the data and use one portion to train our model and use the other portion to test out our model's accuracy. More on this specific implementation is discussed in the Methodology section.

An additional step that we pursued was evaluating how our training set of video game data would work in predicting the sentiment of a data set that was not of video game data. To achieve this, we decided to also test our model on Amazon Musical Instrument data. Results analysis and learning outcomes will be discussed in evaluation section.

For this project, we used Amazon Review Data[1] generously provided by Dr. Julian McAuley of the Computer Science Department at the University of California, San Diego.

**2. Related Work**

The following related work are sources that we did not explicitly use in this project but served as references for our understanding of the various models and algorithms used in this project:

1. http://cs229.stanford.edu/proj2009/AgarwalBhand.pdf

- Sample report on classifying Amazon reviews

2. http://ijcsit.com/docs/Volume%206/vol6issue06/ijcsit2015060652.pdf
   - Sample report on sentiment analysis on Amazon reviews
3. http://ataspinar.com/2016/01/21/sentiment-analysis-with-bag-of-words/
   - Using the bag of words model for sentiment analysis
4. http://pandas.pydata.org/pandas-docs/version/0.15.2/tutorials.html#practical-data-analysis-with-python
   - For pandas tutorials
5. https://web.stanford.edu/~jurafsky/slp3/6.pdf
   - How to classify data well using sentiment analysis
6. http://acl2014.org/CallforPapers.htm
   - how to write ACL style paper
7. http://i.stanford.edu/~julian/pdfs/www13.pdf
   - sample term paper format suggested by professor


**3. Methodology**

The first step that we completed to implement our model was to clean the data we were using for both of our test and training data sets. First, we cleaned up our review text column by removing punctuation, regular expressions, stop-word removal and apply stemming to keep only complete words (implemented in process_text). In this step, we performed feature-reducing methods in order to make our models perform better. Stop-word removal is the process of removing words like “a”, “the”, “and”, etc. These words give us no insight into the sentiment of a revue and thus result in unnecessary features. Stemming is the process of reducing every word that shows up in a review to its root form. For example, the words “argue”, “argued”, “arguing”, and “argument” all become “argu” when stemming is applied by using porter stemmer.[3] This step greatly reduces the number of features for our data sets.

To demonstrate the actions of these feature-removing steps, here is a sample review that was in one of our data sets:

“The product does exactly as it should”

Here is that same sentence but with stop-word removal and stemming:

“product exactli”

In this example, the number of features from the original sentence was reduced by 5.

As helpful as these feature-reducing steps were, they were resulting in very computationally and time-intensive tasks. Our process of stopword removal and stemming was resulting in a very long process that made it tough to perform multiple runs of our code. Our solution to this was to create a separate program to handle the data munging process and save the resulting data into a separate file. This file would then be

passed into our algorithms directly through a different program and therefore we could significantly reduce the time spent in running our main code. This had the added effect of making our code more coherent and concise.

This data munging process was the most intensive part of the entire assignment but it ensured a smooth implementation of our algorithms.

Second, we had to remove numerous columns from our data sets because they were not needed for our purposes.

Here is a list of every column we removed, as well as the justification for its removal:

- Asin
    - Because the data set we used for this project contains information on multiple products, the assignment ID is not helpful
- Helpful
    - Displays how helpful other viewers of the product page found the specific review (i.e. reviews of the review)
- reviewTime
    - Time of the review isn't needed for our purposes
- reviewerID
    - Unique ID for reviewers. Irrelevant for sentiment analysis
- reviewerName
    - Reviewer name. Irrelevant for sentiment analysis
- Summary
    - Bold text prefacing the review. We preferred to look at the contents of the reviews themselves
- unixReviewTime
    - Unix version of review time isn't needed for our purposes

After the removal of these columns, we decided to add a column to the data called “good”. This column, used specifically for our Logistic Regression implementation, was designed to tell us whether a review had a binary evaluation of “good” (1) or “bad” (0). This was determined by examining the overall star rating given by the reviewer. If the reviewer gave the product a 4 or 5 star rating, it would be classified as a “good” review, while a review of 1, 2, or 3 stars would be classified as a “bad” review.

Here is what our data sets looked like after the removal of the unnecessary columns and the addition of the “good” column:

| | overall | reviewText | good |
|---|---|---|---|
| 0 | 5 | Not much to write about here, but it does exac... | 1 |
| 1 | 5 | The product does exactly as it should and is q... | 1 |
| 2 | 5 | The primary job of this device is to block the... | 1 |
| 3 | 5 | Nice windscreen protects my MXL mic and preven... | 1 |
| 4 | 5 | This pop filter is great. It looks and perform... | 1 |
| 5 | 5 | So good that I bought another one. Love the h... | 1 |
| 6 | 5 | I have used monster cables for years, and with... | 1 |
| 7 | 3 | I now use this cable to run from the output of... | 0 |
| 8 | 5 | Perfect for my Epiphone Sheraton II. Monster ... | 1 |
| 9 | 5 | Monster makes the best cables and a lifetime w... | 1 |

We then implemented both Logistic Regression and Naive Bayes' Classifier and began comparing our results. More details on the results are available in the Evaluation section.

**3a. Logistic Regression**

We used the Logistic Regression model found in scikit-learn's linear_model.LogisticRegression,[4] which could be understood mathematically as following:

Logistic regression is a linear model for classification rather than regression which is known as maximum-entropy classification[4] which uses logistic sigmoid function below:

$$g(z) = \frac{1}{1+e^{-z}}$$

Once we use that to find the hypothesis function $h_\theta$ for given x parameterized over $\theta$ to predict probability y, based on $\theta^T x$ which can be expressed as:

$$h_\theta(x) = g(\theta^T x)$$

Since the sigmoid function has special properties that can result in values in the range [0,1], if $h_\theta$ is greater than or equal to 0.5 then we can predict y = 1, and y = 0 when it's less than 0.5. Using this as classification of yes/no, we found the likelihood of a "good" or "bad" review for our featured data.

The next goal of logistic regression is to minimize this cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Based on the hypothesis function result, $\theta$ should be optimized in order to give the best possible reproduction training set and maximum likelihood probability. Since our data is not too large to use stochastic gradient descent, we use this gradient descent for logistic regression:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Therefore, the hypothesis function is the key of logistic regression when drawing our decision boundary. We used binary logistic regression, but for multinomial cases, our algorithm can simply extend to $\theta^T x$ as below:

- $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

**3b. Naive Bayes' Classifier**

For the purposes of this project, we used the multinomial implementation of Naive Bayes' Classifier.

Naive Bayes' Classifies in its essence is a modification of Bayes' theorem of conditional probability.We can get the probabilities by looking at the likelihood of prior probabilities, as shown below.

1. Get maximum posterior probability P(x|y) of given class $c \in C$ by substituting these two formulas

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

By following these two steps can we get the estimate of the correct class:

3. $$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)}$$

4. Therefore, computing feature *d* by choosing the class which has the highest product of two probabilities P(c) and P(d|c) gives us:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

However, we would still require a large number of parameters and impossibly large training sets to estimate the probability of every possible combination of features when we have more than one.

Naive Bayes' Classifier therefore makes two simplifying assumptions.The first is the bag of words assumption and the second is the naive bayes' assumption using conditional probability which we discussed above.

Then, for the general case, using features $f_1$ $f_2$ $f_3$ ..$f_n$ $\in$ d for the final equation for the class chosen by a Naive Bayes' Classifier would be:

$$P(f_1, f_2, ...., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_n|c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

This is also known as the decision rule. Since the goal is to maximize the decision rule, we take the log of the maximum likelihood and introduce $\theta$ such that:

$$\ln P(f|\theta) = \sum_{i=1}^{n} ln\, P(f_i|\theta)$$

Thanks to polymorphism, all these computations are all done by the python library sklearn.naive_bayes.MultinomialNB[5].

**4. Evaluation**

For evaluating the performance of Logistic Regression in our analysis, we used our previously discussed

categories of whether or not a product was deemed good (1) or bad (0). Using this, we looked at the individual score which our algorithm gave to each product and compared our score of 0 or 1 against the correct score. This way we were able to determine the accuracy of our algorithm as shown in table below.

For evaluation the Naive Bayes Classifier, we compared the score (1-5) that our algorithm determined for each data point and compared it with the real value. Since there are 5 different values that the algorithm could potentially give for each product, the margin for error was much greater than it was for the Logistic Regression algorithm, as that algorithm only had 2 values (0 or 1). However, these results came out more precise.

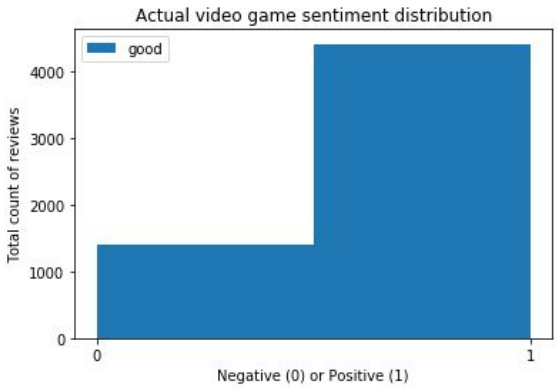
Here is a table of the distribution of ratings from our test data:

| Train on Game data | Game data Test | alpha adjustment for NB | Music data Test |
|---|---|---|---|
| L.R | 0.96 | N/A | 0.68 |
| N.B | 0.69 | a = 1 | 0.79 |

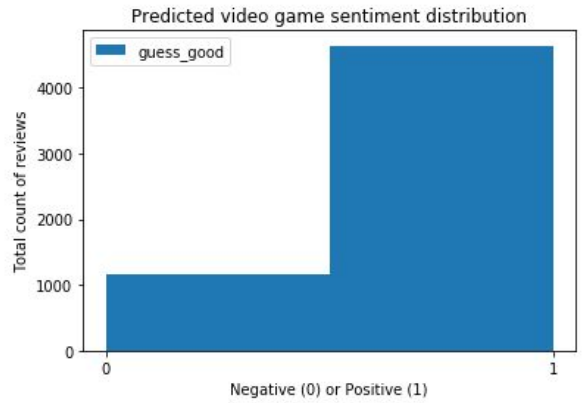
Logistic Regression Visualizations

Recall that we used Amazon video game data for our training data.

Here is the true distribution of the Amazon video game test data using the good/bad scale we created for Logistic Regression:

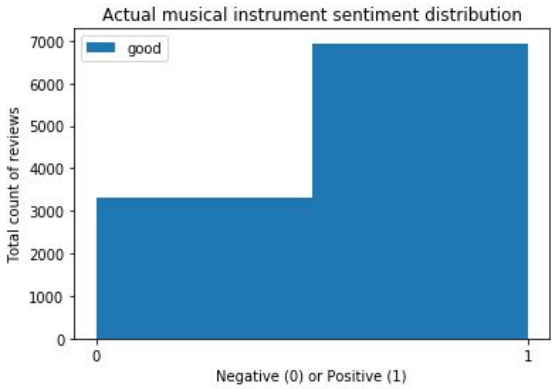



This graph shows that the majority of the reviews in the test data set are positive reviews while our Logistic Regression model predicted the distribution for the Amazon video game test data:



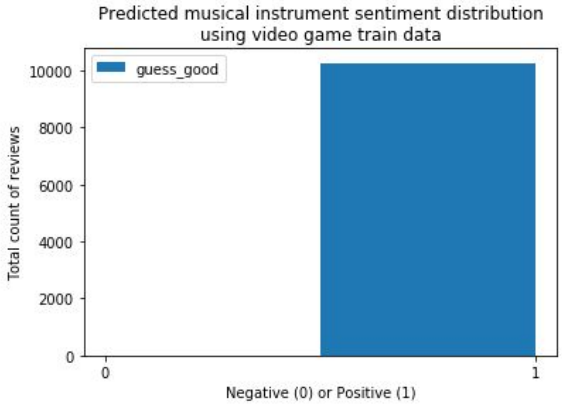

Our model follows the true distribution of the test data set by giving us a majority of positive reviews. This gives slightly more positive reviews and slightly fewer negative reviews but for the most part, this model seems accurate.

Here is the true distribution of the Amazon musical instrument test data using the good/bad scale we created for Logistic Regression:




Similar to our video game test data set, this musical instrument data set also has a majority of positive reviews.

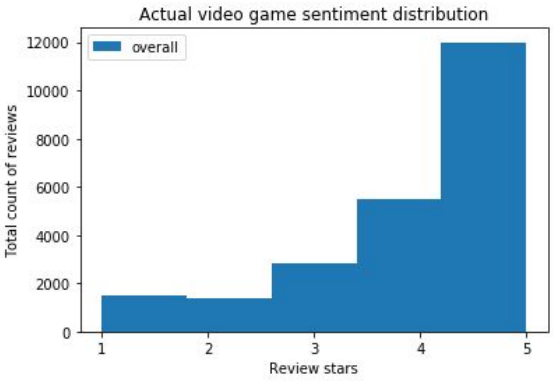
This is how our Logistic Regression model predicted the distribution for the Amazon musical instrument test data:




This graph is truly perplexing because our Logistic Regression model when tested on a data set of information that is unrelated to the test data set does very poorly. Our model was not able to recognize a single review as a negative review.

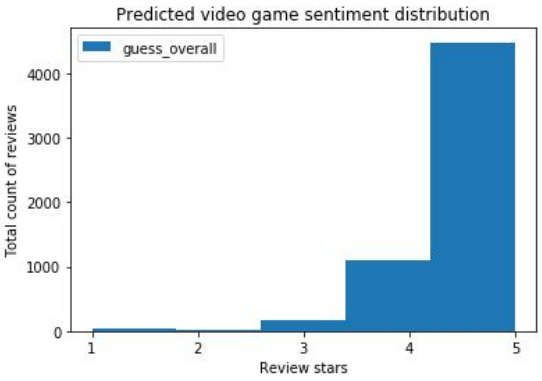
Naive Bayes' Classifier Visualizations

Here is the true distribution of the Amazon video game test data using Amazon's 1-5 star scale:




We see a fairly even distribution of 1, 2, and 3 star reviews but there are clearly more 4 and 5 star reviews.

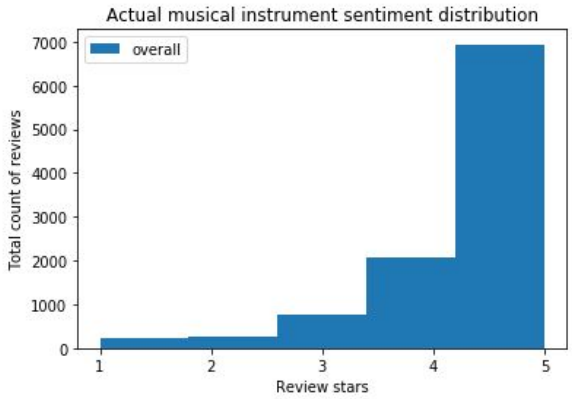
This is how our Naive Bayes' Classifier model predicted the distribution for the Amazon video game test data:




The shape of our distribution is similar to the true distribution but the model still

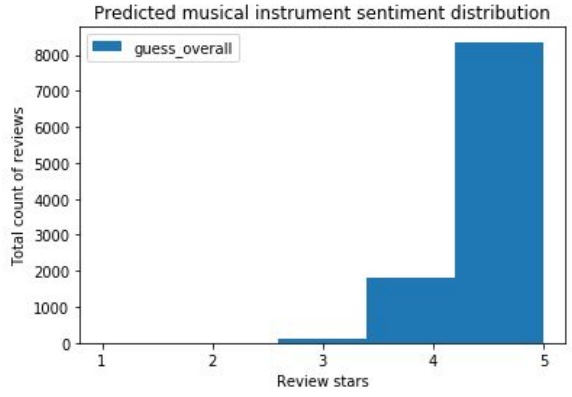
doesn't identify too many 1, 2, or 3 star reviews.

Here is the true distribution of the Amazon musical instrument test data using Amazon's 1-5 star scale:




This data is less spread out than the video game data set but just like that data set, it skews towards 4 and 5 star reviews.

This is how our Naive Bayes' Classifier model predicted the distribution for the Amazon musical instrument test data:

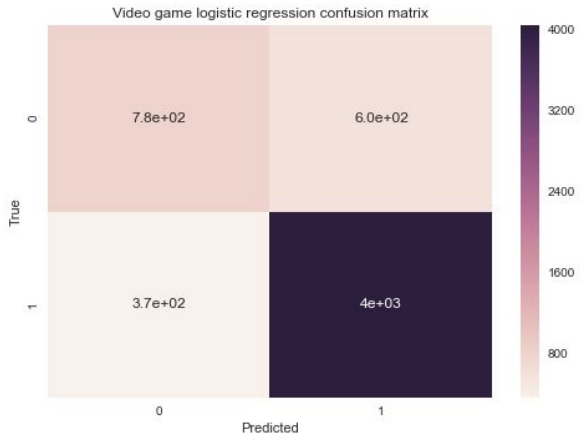



Because there were few 1, 2, or 3 star reviews in the true distribution, this model performs very well. The shape of the distribution looks very similar to the true shape.

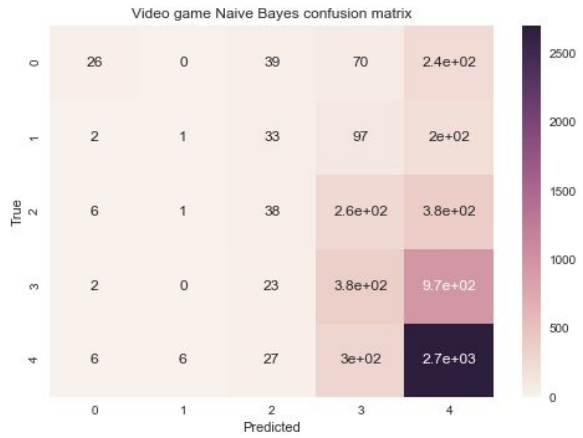
Confusion Matrices

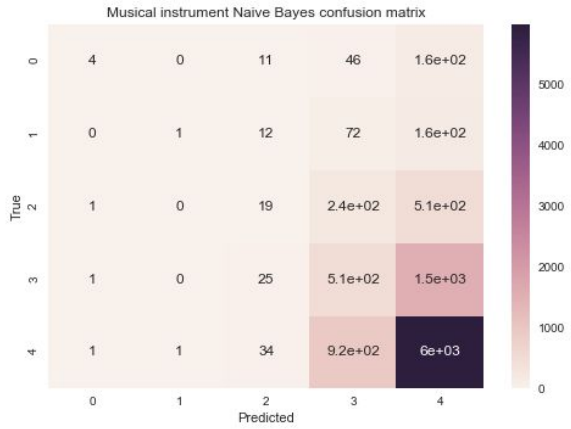
Here are confusion matrices for our data. They show the number of correct and incorrect predictions made by our models. They also display false positives and false negatives, which is informative as to the possible shortcomings of each algorithm.

Confusion Matrix for Amazon Video Game test data set using Logistic Regression:

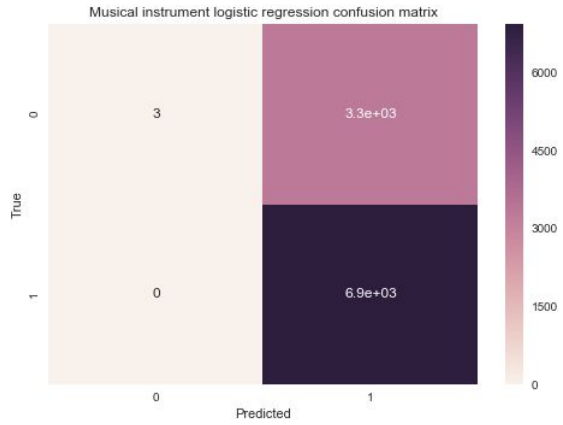



Confusion Matrix for Amazon Video Game test data set using Naive Bayes' Classifier:




Confusion Matrix for Amazon Musical Instrument test data set using Logistic Regression:




Confusion Matrix for Amazon Musical Instrument test data set using Naive Bayes' Classifier:

## 5. Conclusion

This was a very rewarding experience as it was our first data science project. We got to learn many machine learning techniques and were able to use them to try and solve real world problems. It was a rewarding experience and we can certainly see how this sort of work is very useful in today's information age.

As mentioned in the introduction, Machine Learning is one of the great solutions for classifying good and bad reviews. We found that Logistic Regression is better-suited for this task than Naive Bayes' Classifier.

One implementation challenge we faced with this project was with the way we implemented our code. Specifically, we used Jupyter Notebook for our project because of its inherent benefits for data science projects. One big drawback with the Notebook is that there is no way to collaborate in real time on the code. Because the file extension .ipynb is not

very common, Github had difficulty merging changes and commits which resulted in a lot of changes having to be manually inserted into our individual notebooks after they were made.

For future iterations of this project, it would be wise to use a platform, such as Google Cloud Platform, to allow us to use much larger data sets in our testing. This will allow us to get more accurate results. This would certainly be more helpful for larger scale projects.

Another aspect that could be improved upon is improving our test data set. We faced a tricky situation in that the majority of our test data skewed towards positive reviews which meant that our model was quite poor in detecting bad reviews. In the future, with the help of a much larger test data set, we can ensure that there is more of an equal distribution of positive and negative reviews.

Through our experiments with multiple data sets, we came to the conclusion that Naive Bayes' Classifier is a much better algorithm for handling data from an unfamiliar source. In our testing of the musical instrument data against the video game data set, our Naive Bayes' Classifier implementation handled the data set quite well. Logistic Regression, however, was terrible at determining the sentiment of the musical instrument data. We did notice that Logistic Regression performed better than Naive Bayes' Classifier when the training and testing data sets were both of video game reviews. This led us to the conclusion that Logistic Regression works significantly better when the training and test data come from a similar dataset. In a real-world setting, the training and testing data sets are more likely to be from of a similar type, which makes Logistic Regression the preferred algorithm for performing sentiment analysis.

**References**


1. http://jmcauley.ucsd.edu/data/amazon/links.html
2. https://pypi.python.org/pypi/stop-words
3. http://www.nltk.org/api/nltk.stem.html
4. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5. http://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes


**Appendix A: Work Distributions**

- Thida: In charge of Naive Bayes' Classifier, documentation lead
- Sanjay: In charge of Logistic Regression, project manager
- Erik: In charge of Data Munging, programming lead

**Appendix B: Setup Instruction**

The following instructions are included in the README.md file attached to the codebase located at

https://github.com/sanjaykaliyur/CSCI183-Project:

1. Clone this project using git clone git@github.com:sanjaykaliyur/CSCI183-Project.git, or download the zip file from the github page.
2. If you don't already have it, install Python 3.5+ from python.org or using sudo apt-get install python3.
3. Install pip3
    - Unix: sudo apt-get install python3-pip.
    - Other: Follow pip's manual installation instructions.
4. Ensure pip is up to date
    - pip3 install --upgrade pip
5. Install Jupyter
    - pip3 install jupyter
6. Install required dependencies
    - pip3 install $(cat install-requires.txt)
7. Extract the compressed data files
    - tar -xvf data.tar.gz
8. Open the project notebook
    - python3 -m jupyter notebook
    - Sentiment-analysis--amazon_commented.ipynb

External Dependencies

matplotlib: Data plotting tool
nltk: Natural language toolkit, used for stemming
pandas: Data processing toolkits
sklearn: Machine learning algorithm library
seaborn: Used for the heap-map confusion matrices.