Sentiment Classification of Amazon Book Reviews
Springboard DSCT Capstone Project I

By: Mindy Ng

August, 2017

The Problem†

There has been a deluge of what people are thinking on the World Wide Web. The challenge is organizing all of it in a way so that there can be business impact.

Consumers voice their opinions on blogs, Tweets, Facebook comments, Pintrest tags, SnapChat's, Instagram posts, YouTube comments and Google+ comments. It would be valuable to organize these thoughts for businesses to make smarter decisions.

In general, people rely heavily on others' opinions in order to make purchase decisions. Here are some statistics from a paper called, "Opinion Mining and Sentiment Analysis" by Bo Pang and Lillian Lee.

- "Among readers of online reviews of restaurants, hotels, and various services (e.g., travel agencies or doctors), between 73% and 87% report that reviews had a significant influence on their purchase."
- "Consumers report being willing to pay from 20% to 99% more for a 5-star-rated item than a 4-star-rated item (the variance stems from what type of item or service is considered.)"

While consumers can easily read through many reviews in order to get a better idea of a product of interest, classifying reviews into a few categories such as positive, neutral and negative would help make the decision process easier on the business end.

Online sentiments would be of use to businesses that would like to know preliminary opinions on new or existing products, such as a cell phone that just launched onto the market. In order to gauge the amount of favoritism or dislike, a business could use a data analysis model, like the one I am building in this capstone project, to survey a large amount of users' responses to the product. This would enable the business to get a quick idea of the product's future success rate. Therefore, this would provide insight into whether or not the product would be a good business investment.

My approach is to perform Supervised Learning on reviews associated with 8 different books, for a total of 243,269 book reviews, which have been obtained from UCI's Machine Learning Repository. A portion will be used as the training set and the other portion will be used as the test set. Then performance metrics will be reported on three different Supervised Learning algorithms to gauge how well it performed in predicting a review's sentiment label.

The Dataset†

All datasets were in .csv file format. The information that the dataset has is a collection of reviews from 8 different books. For each review, there are four columns, but only two are used for this capstone project. The relevant columns/important fields are “Review Score” and “HTML of Review Text”. The “Review Score” column has the product rating on a scale of 1-5 and the “HTML of Review Text” has the review text.

These were used to quantify a review’s sentiment: negative, neutral or positive.

Some Caveats About this Dataset

A book that is highly disliked is not represented well in this dataset, which makes the dataset limited to some extent. Therefore, a new negative review processed in an unsupervised machine learning algorithm would have a low probability of being classified in its correct category if the algorithm only had a few negative reviews to be trained on.

If this dataset were used for sentiment analysis, this dataset comes with the usual difficulties associated with trying to discern a person’s intent based on text alone. Subtleties/nuances in meaning may not be caught, e.g. when sarcasm is used. If sarcasm is used, then a straight literal interpretation of someone’s review would not correctly represent person’s sentiment. Sarcasm typically uses constructions that look like the opposite of what the person is actually trying to say.

Justifications for Using Datasets

Though the trained unsupervised machine learning algorithm would not be prepared to classify a negative review, it is rare that people take the time to write a review on something they do not like.

Data Wrangling†

The very first step after downloading and unzipping the dataset was to import all 8 separate CSV files and format them as individual Pandas data frames. Each data frame had a review per row. Each data frame had 4 different columns (from left to right): “Review Score”, “Tail of Review URL”, “Review Title” and “Review Text”.

All reviews were combined into one big dataframe to make data wrangling easier- such as applying functions on it.

Then columns: “Review Score” and “Review Text” were separated out as their own variables since these would be the main objects handled in the Machine Learning algorithms.

Given that each “Review Text” had HTML tags, pre-processing was done on the text in order to prepare it for use in the Machine Learning algorithms. Since each entry in the “Review Text” column was a string object, string methods were applied in order to strip, replace, and translate the string to get it into pure text form - no HTML tags nor punctuation.

Each processed review string was then recombined into one big list with all reviews from all books.

Once the reviews were in a suitable form, such as not having any HTML tags nor punctuation, they were vectorized into a dictionary. This means that each review was made into a vector with dictionary key-value pairs. The keys were the words in the review and the values were the word counts. This was made into the value X that would eventually be fed into the first Machine Learning classifier - MultiNomial Naive Bayes classifier.

All vectorized reviews were arranged in a matrix (which I called “X” in my code) such that its number of rows was the number of reviews and its number of columns was the number of words in the dictionary built by “CountVectorizer”. For review in row i of matrix X, X[i, j] = 1 if and only if word in position i in the dictionary occurred in the review, and X[i, j] = 0 otherwise.

After X was defined, the review scores were labeled depending on their score. Reviews with scores between 1 and 2 were labeled as 1 for “negative”, reviews with score equal to 3 were labeled as 2 for “neutral”; and reviews with scores between 4 and 5 were labeled as 3 for “positive”. Vector y was defined using these labels such that y[i] was set to be equal to the label of the review. The pair (X, y) thus defined a multi-class classification problem, which could be tackled using a variety of algorithms. In this project, I tried the following: MultiNomial Naive Bayes, Decision Trees, and Random Forests.

Perform Classification using MultiNomial Naive Bayes Classification Model

MultiNomial Naive Bayes classifier was chosen because it is “suitable for classification with discrete features (e.g., word counts for text classification)” according to scikit learn’s documentation on the MultiNomial Naive Bayes classifier. Also, it “can be trained very efficiently in a supervised learning setting” and “despite [its] naive design and apparently oversimplified assumptions, Naive Bayes classifier has worked quite well in many complex real-world situations” according to Wikipedia.org. And in plain English, with the Naive Bayes theorem, “we are computing the probability that a document (or whatever we are classifying)

belongs to category c given the features in the document...the particular Naive Bayes classifier that we are using is Multinomial Naive Bayes since "we are modeling word counts" as is noted by the Naive Bayes Mini-Project author.

In order to perform classification using this sci-kit learn's machine learning algorithm, the dataset was split into training and test sets. After this, an accuracy percentage was determined on both training and test sets. Accuracy was chosen as one of the model performance metrics because as Dr. Jason Brownlee wrote on his *Machine Learning Mastery* blog, "model performance is estimated in terms of accuracy to predict occurrence of an event on unseen data. A more accurate model is seen as a more valuable model. A model with higher the accuracy can mean more opportunities, benefits, time or money to a company. And as such prediction accuracy is optimized."

After running the multinomial Naive Bayes model, the accuracy percentages were 0.75 for the training set and 0.72 for the test set. This was not bad considering that they both are close in values. This means that the model did not overfit. Thus, the performance of this classifier was not bad. It did as expected from the documentation which said that multinomial Naive Bayes works quite well. Other model performance metrics I used were Precision, Recall and F1-Score. Precision is defined as the accuracy of positive predictions. Recall is intuitively the ability of the classifier to find all the positive samples. And F1-Score is the weighted average of the Precision and Recall, where an F1-Score reaches its best value at 1 and worst score at 0. The relative contribution of Precision and Recall to the F1 score are equal. Thus, a good classification model would have all metrics- Precision, Recall and F1-Score as close to 1.0 as possible. When looking at multinomial Naive Bayes model's results, it was clear that the positive sentiments category (#3) scored highest due to the text corpus having mostly positive reviews. Thus, a model would be able to predict well on data that it was trained mostly on. Similar results with category 3 scoring highest were seen after running the other classifiers - Decision Tree and Random Forests.

Perform Classification using Decision Tree Classification Model

Decision Tree Classifier was chosen because according to Aurelien Geron in Hands-On Machine Learning with Scikit-Learn and TensorFlow they are capable of fitting complex datasets. This model is also the fundamental component of Random Forests, which are among the most powerful Machine Learning algorithms available today." It works by starting at a root node (depth 0) which asks a question based on a feature from the model's feature set. Based on the answer to the question, the branch moves down to the left or the right child node (depth 1). And this goes on until the tree's max depth is reached, which is specified as a hyperparameter when setting up how to run the Decision Tree model. When the max depth is reached, a classification decision is made.

After running the Decision Tree model, the accuracy percentages were 0.99 for the training set and 0.63 for the test set. This was not bad considering that the training set accuracy is close to 1.0. This means that the model almost made perfect predictions on the training data. One improvement to this would be to change the max_depth parameter in the Decision Tree model set up so that it would not go so low in the tree (set max_depth to lower number). This would enable the model to be more general. And thus, the difference between its Training and Test set accuracies would not be as great. The greater the max_depth setting tends to constrict the model further than needed, making a fit on unseen data very difficult. Even though this was the outcome from this model, there is another one that is available which might produce higher accuracy values.

Perform Classification using Random Forest Classification Model

Random Forest Classifier was another chosen model because it is an ensemble of Decision Trees. An ensemble method is a group of predictors, which is advantageous. And as Aurelien Geron in Hands-On Machine Learning with Scikit-Learn and TensorFlow said, "If you aggregate the predictions of a group of predictors, you will often get better predictions than with the best individual predictor. Despite its simplicity, this is one of the most powerful Machine Learning algorithms today." Straight from the same reference book: "They work by being trained via the bagging method typically with max_samples set to the size of the training set. Instead of building a BaggingClassifier and passing it a DecisionTreeClassifier, you can instead use the RandomForestClassifier class, which is more convenient and optimized for Decision Trees. With a few exceptions, a RandomForestClassifier has has all the hyperparameters of a DecisionTreeClassifier, plus all the hyperparameters of a BaggingClassifier to control the ensemble itself. The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity, which trades a higher bias for a lower variance, generally yielding an overall better model."

After running the Random Forest model, the accuracy percentages were 0.97 for the training set and 0.72 for the test set. 0.97 was close to 1.0, which meant that the predictions were close to perfect. At the same time, the accuracy value for the test set was much lower, with a difference of 0.25 between the training and test sets. This means the model is overfitting the training data. The gap could have been smaller if the tree was not set on its max_depth default setting. The tree's training must have gone fairly deep for it to not generalize enough to fit well when it was introduced to new test data. Despite being run on default settings, the Random Forest performed the best compared to the MultiNomial Naive Bayes and Decision Tree models. The separation between the training set and test set accuracies was not as big as the separation observed for the

other models. And the test set accuracy value was the highest out of all the models run on the data.

Recommended Use of Sentiment Classification†

Though the reviews were already pre-labeled with a review score, it is hard at times to determine efficiently which reviews are negative, neutral or positive with a massive amount of text to look at all at once. This is why having a trained text classifier would help businesses swiftly sift through a corpus of text to determine the amount of favorability or dislike towards a product. This would save on time and costs when in the long run there will be constant opinions made on what people buy. So the amount of customer reviews to analyze would be a challenge, but less so with this classifier.

Concrete Recommendations On How to Use My Findings†

a. Use sentiment classifier to review other Amazon products besides books, such as movies, beauty products, clothes, kitchen appliances.
b. Use sentiment classifier on Twitter tweets after a product is launched to get an initial sampling of customer receptivity.
c. Use sentiment classifier as social guage during sports games to see how fans are reacting towards a player. This would aid in sports companies determining which athletes they would want to sponsor in an effort to increase their customer base.
d. Use sentiment classifier on newswires in order to determine high profile media personalities' approval ratings, which would lead to business impact.

Potential Next Steps†

a. Remove stop words, and perform "stemming" in order to reduce dictionary dimensionality so that model runs more efficiently (for instance, by seeing the overall execution time being diminished.)
b. Tune model hyperparameters to use optimal settings, for instance by using grid search.
c. Use more complex ensemble methods (besides Random Forests) to produce more accurate classification models.
d. Perform sentiment analysis which would involve using semantics and syntax to dive deeper into text corpus to find nuances in text.
e. Perform Unsupervised Machine Learning since not knowing the labels for a review is what is usually encountered in the real world.

Conclusions/Summary†

With a such a deluge of opinions and sentiments across the world wide web from blogs, Twitter tweets, Facebook comments, Pintrest tags, SnapChat's, Instagram posts, YouTube comments and Google+ comments, one huge business need is determining what is the overall sentiment on a product(s). In order to capture this in a swift way, this trained classifier is the tool to use to deliver those highly valuable results.