SENTIMENT CLASSIFICATION USING AMAZON REVIEWS

A SUPERVISED MACHINE LEARNING APPROACH TO PREDICT CUSTOMER RATINGS

GRANT KENNEDY & RICHARD ROBINSON

Contents

Introduction

One of the many unique advantages of online marketplaces is the ability of shoppers to quickly compare items by their review scores. These scores represent the average customer experience of all the customers chose to write a review. Sites like Amazon even let shoppers sort products according to their score, or even query only the items which meet a certain threshold of satisfaction. This has enabled a degree of safety for online shoppers without needing to place any trust in particular brands or sellers. Amazon shoppers can read any particular review on a product, but it is often sufficient to chose a product based on its star rating.

However, there are many other uses for sentiment analysis where quantitative scores may not be available. Reddit, Twitter, and YouTube comment sections are examples of forums where users have no way of measuring average sentiment about a particular topic or product.

This project implements a supervised machine learning model to predict customer sentiment based on Amazon review text.

How to Run this Project

This project was developed within the Anaconda programming environment for Python 3.7.1, this environment comes with all the dependencies needed to run this project.

Enter “python run.py” in the Anaconda Prompt from the root of this project folder. The current configuration should produce scoring metrics in about 5 seconds depending on your machine. The results and configuration are then written to a new row in “Results/Amazon_Review_Results.CSV” and the confusion matrix is plotted to the screen.

The dataset path, classifier, features, and other configurations can be changed by hand in the “configuration.py” file.

Related Work

There is already a fairly large body of related work already on this topic. In preparation for this project we read a number of academic papers listed here. Many of them used Amazon reviews to classify text as either positive or negative -- binary classification. We chose to build a model which would attempt to predict the actual score -- out of 5-stars -- of a particular review.

Sentiment Classification on Amazon Reviews using Machine Learning Approaches, Sepideh Paknejad: Kth Royal Institute of Technology, Stockholm Sweden

Predicting Rating of Amazon reviews - Techniques for Imbalanced Datasets, Marie Martin: University of Liège

Amazon Rating Prediction, Peter Brydon, Kevin Groarke – UCSD

Methodology

Data setup

The Amazon Customer Reviews Dataset is publicly available and can be accessed through the Amazon AWS infrastructure. The dataset is segmented by product category, and each category of data is available for download as a compressed Tab Separated Values (TSV) file. At the time of writing, the video games dataset we used was available for download at the following link:

```
https://s3.amazonaws.com/amazon-reviews-
pds/tsv/amazon_reviews_us_Video_Games_v1_00.tsv.gz
```

The dataset was substantial: a 1.1GB TSV file comprising 1,786,030 customer reviews from November 1997 until August 2015. We truncated the dataset to the most recent 100,000 reviews which contained reviews from only two months in 2015.

In order to take advantage of the Scikit-learn machine learning tool set, we used Python 3.7.1, and worked within the Anaconda programming environment because it ships with the latest Scikit-learn library. Because Python has better support for Comma Separated Value (CSV) format, we made a python script file_conversion.py to convert the TSV file into CSV format.

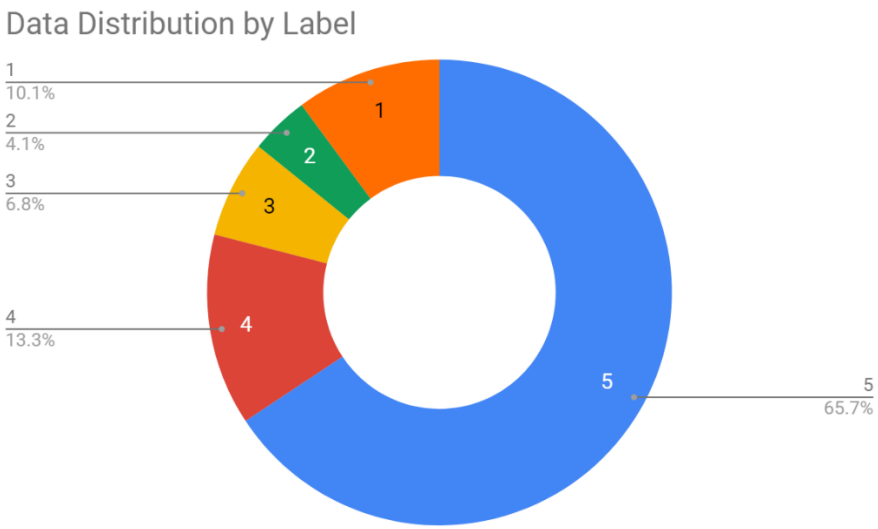
The following data fields were discarded:

~~marketplace~~ ~~customer_id~~ ~~review_id~~ ~~product_id~~ ~~product_parent~~ ~~product_title~~ ~~product_category~~ star_rating helpful_votes total_votes vine verified_purchase review_headline review_body ~~review_date~~

The star rating field contains the labels for our supervised model, an integer between 1-5

Resampling the data

Customer reviews tend to be heavily imbalanced. To illustrate the imbalance of our dataset, we sorted the first 10,000 reviews by label:



Our model must be able to predict the labels of customer reviews. However, with the above distribution of labels, the model could classify all reviews as 5-stars and have a score above 65%. To account for this, we applied the random under sampling transformation on our data. This function randomly removes items from the majority classes to match the number of items in the minority class, which in our data was the 2-star class.

This way our model will have the same number of items from each class to train and test with. The consequence is that the items chosen from majority classes depend completely on the pseudo random state of the Random Under Sampling method, whereas the items of the minority class are not as affected since nearly all of its items remain after this transformation.

Feature Extraction

Bag of Words

The first step of feature extraction was to apply the bag of words model on both the body and headline text fields. This was done using either the Count Vectorizer or Tfidf Vectorizer. The CountVectorizer converts a collection of data into a sparse matrix of token counts, whereas the TfidfVectorizer does the same but with the tfidf values for each token. The length of the sparse matrix will represent the length of the dictionary of tokens found in the collection.

We chose to use the Count Vectorizer for most of our tests and could not find a definitive advantage of one over the other. This method allows the removal of a standard list of English stop words. Stop words are English words such as “it” or “and” which do not help convey sentiment and increase the runtime of training considerably because of their prevalence. This method also allows the specification of n-gram range, which is discussed later.

Custom Features

Many features of language and sentiment are left un-extracted by the bag of words model. For instance, the use of three periods such as “it’s okay…” is a linguistic feature that can express hesitancy, sarcasm, or disappointment. As another example, the use of all capital letters and exclamation marks such as “THIS SUCKS!!!!!” are other human recognizable expressions of emotional ‘volume’ which are missed by a simple bag of words model. Our model should consider “THIS SUCKS!!!!!” as more negative than “this sucks” despite having the same bag of words. Therefore we recognized the need to implement additional methods to extract these features.

We created another feature extraction vectorizer named Function Featurizer. This method implements a sci-kit recognizable transformer based on simple functions which analyze the text and return some value. For example, one function “capitalizationRatio” will return the ratio of capitalized letters in the given text. This enabled us to easily write several feature extractors based on simple functions.

We included the following custom features in our model:

- capitalizationRatio -- returns the ratio of capitalized letters
- Exclamation – returns the number of exclamation marks
- Question – returns the number of question marks
- Dots – returns the number of instances of "…"
- Length – returns the length of the text
- Emojis – returns 0 if more sad emojis, 2 if more happy emojis, 1 if none found (":D", "D:", "(:", ":(", etc.)

Feature Union

One way of extracting features from both the body and the headline is to simply merge the two fields into one. However, this method prevents the extraction of some potential features such as the length of the headline, or the capitalization ratio of the headline itself. Further, we considered that while removing stop words from the body would greatly reduce running time, it may be overly punitive on the headline which is usually only a few words. For example, the headline "It is what it is" happens to be made entirely of English stop words. For these reasons we recognized the need to extract different features from each field. We created two separate Feature Unions named BodyUnion and HeadlineUnion. The Scikit-learn Feature Union is a method of applying multiple feature extractors sequentially on a text and producing a single vector representing each document.

A Column Transformer is used to apply the selected features onto selected columns of the data and extend each set of vectors column wise into one sparse matrix.

The Column Transformer then transforms the data for the training phase.

Training

Training and cross validation is handled by the sklearn cross_validate method. The number of folds is specified, as well as our scoring metrics to be returned. We specify Precision, Recall, and F1.

Results are printed after runtime completes, and two optional methods can record the results and current configuration to a CSV file and plot the resulting confusion matrix.

Results

Headline vs. Body

We compared results for training with unigrams on the Body only vs. the Headline only. Comparing f1 scores, we see that the Headline accounted for most of the correctly classified documents. Considering the Body as well as the Headline only contributed 4% to the score, indicating a large overlap between the sets of documents which each union helped to classify.

Body Only

	Precision	Recall	f1
SVC	.442	.438	**.411**
NB	.442	.438	**.436**

Headline Only

	Precision	Recall	f1
SVC	.638	.596	**.601**
NB	.61	.611	**.61**

Combined

	Precision	Recall	f1
SVC	.645	.647	**.644**
NB	.657	.607	**.618**

CV = 5, data size = 30k before resampling

N-gram range

This figure shows preliminary f1 scores comparing the use of unigrams and bigrams in feature extraction. Bigrams improved performance over 1% with SVC and 8% with NB.

	Unigrams	Bigrams
SVC	.654	**.667**
NB	.571	**.65**

data size = 90k before resampling, no CV

Stop words

Despite some early results which suggested stop words should *not* be removed, we found best results by removing stop words from the body text. performance improved 0.6% with SVC. Performance decreased with NB, but this may be an outlier result for this configuration, since we have seen a similar increase for NB with larger data sizes.

	control	stopwords removed
SVC	.66	**.666**
NB	.64	**.632**

data size = 60k before resampling, no CV

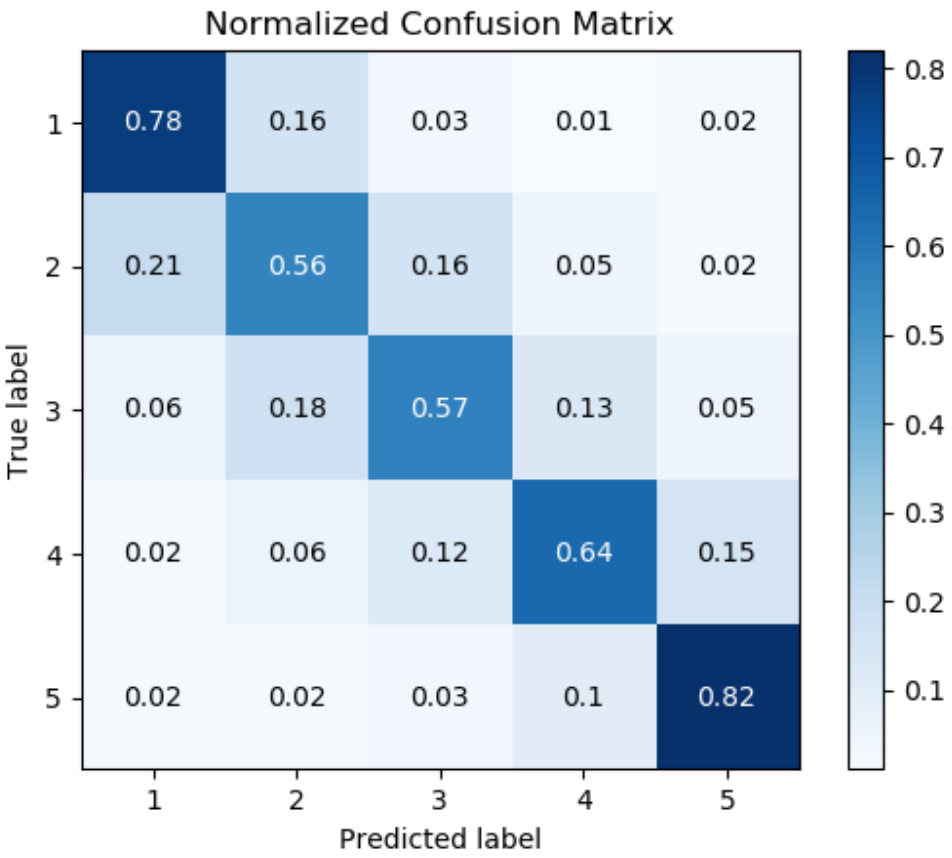
Best Results

The following results represent the best-case performance we observed from our model. The experiment was run on the 90k reviews dataset, 10-fold cross validation with both SVC and Naïve Bayes classifiers. The difference in runtime is considerable; NB performed within 0.4% of SVC, but finished over 200 times faster.

	Precision	Recall	F1	Runtime (sec)
SVC	0.672	0.674	**0.672**	1035
NB	0.695	0.666	**0.668**	4.8

CV = 10, data size = 90k before resampling, Majority class = 20%

And the confusion matrix for the svc experiment:



We tuned this model to Binary classification mode by converting the labels {5,4,3} → 1 and {2,1} → 0 and got the following scores:

	Precision	Recall	F1
SVC	0.931	0.93	**0.93**
NB	0.923	0.923	**0.923**

CV = 10, data size = 90k before resampling, Majority class = 50%

Our result score for SVC is within 1% of results found in “Sentiment classification on Amazon reviews using machine learning approaches” which achieved binary classification by omitting 3-star reviews from the model.

Analysis

Feature Ablation Study

One way to quantify the increase in performance of our model due to each feature, is to remove features from the model one at a time and compare the results of the model without those features.

The below figure shows the feature ablation study. Each experiment was done on a 40k dataset size, 2-fold Cross Validation.

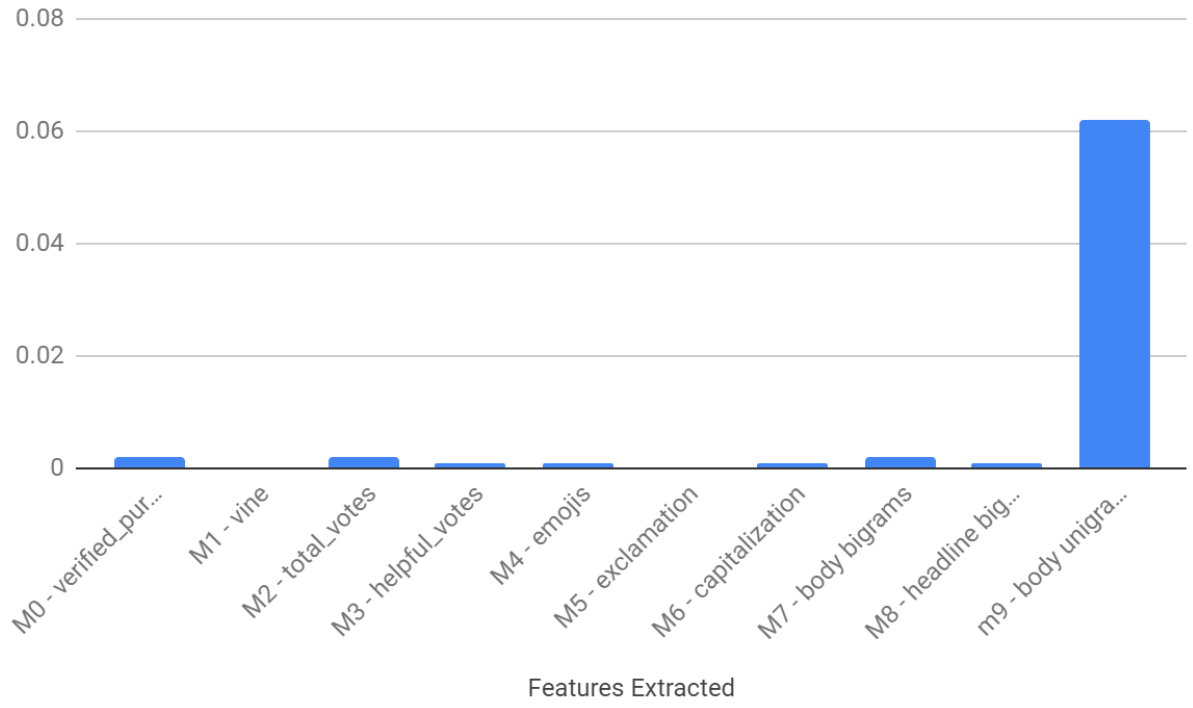
M0 is the fully featured model. One feature is removed in each succeeding row, such that M2 is the model with features of M1 *minus* the vine feature. A larger negative value indicates a better feature. Results rounded to 3 decimal places.

Model	Features Extracted	Score (f1)	Runtime (sec)	Difference after ablation
M0	All Features	0.658	50.4	
M1	M0 - verified_purchase	0.656	51.5	-0.002
M2	M1 - vine	0.656	48.6	0
M3	M2 - total_votes	0.654	48.9	-0.002
M4	M3 - helpful_votes	0.653	49.5	-0.001
M5	M4 - emojis	0.652	49.8	-0.001
M6	M5 - exclamation	0.652	49	0
M7	M6 - capitalization	0.651	45.5	-0.001
M8	M7 - body bigrams	0.649	24.7	-0.002
M9	M8 - headline bigrams	0.648	23.4	-0.001
M10	m9 - body unigrams	0.586	4.6	-0.062

Note that the value added to the model by each feature depends on what features are already present, because overlap exists between the reviews which each model is able to classify. In other words, the order of ablation matters.

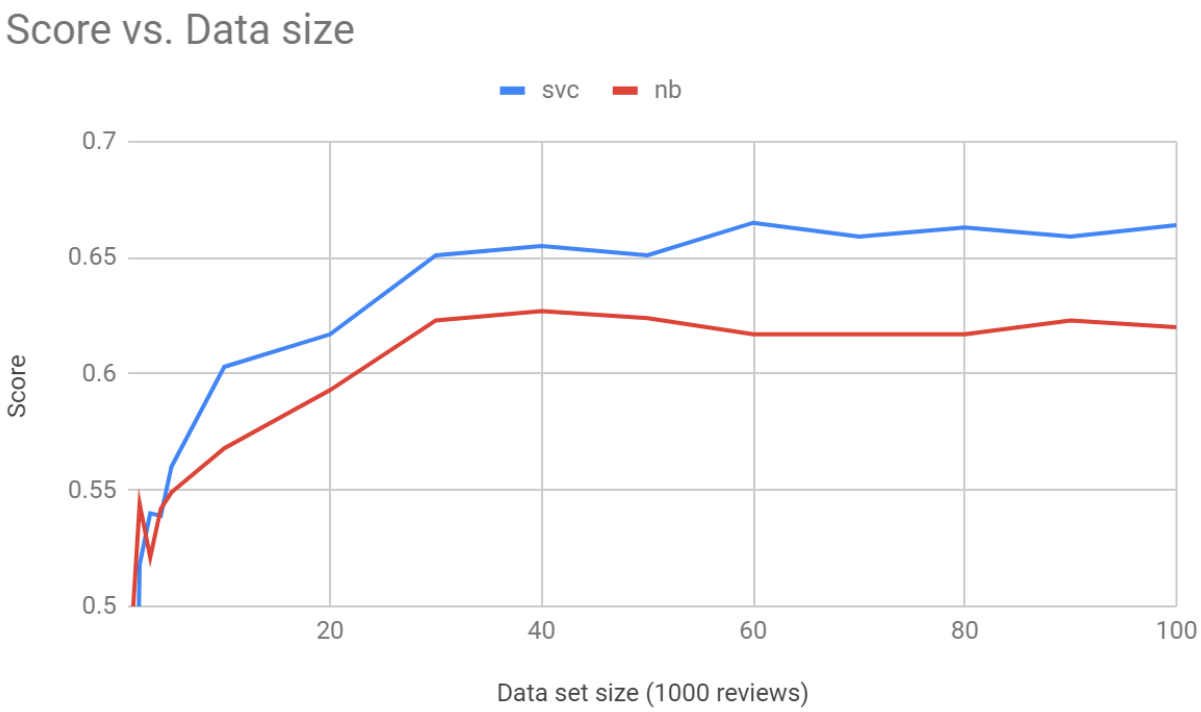
Most custom features accounted for only .001 or less. And the body unigrams only classified an additional 6% of reviews which M10 (headline unigrams only) did not.

Below is a visualization of each feature's relative contribution from the above chart



Comparing Dataset Sizes

Prediction scores are expected to increase with the amount of data used to train a classifier. To illustrate this dependence, we ran successive experiments on several datasets of increasing size to plot performance of SVC and NB vs. data set size:



With smaller datasets, Naïve Bayes out performs the SVC classifier. But SVC outpaces NB in our tests beyond datasets of about 5,000 (before under sampling). This matched our prediction based on related work about the nature of Naïve Bayes and Support Vector Classification.

Uncertainty due to Resampling

Because our model uses random under sampling to shrink each class of data to match the minority class, our model's performance has a considerable amount of dependence on the random state of the random under sampling method. This pseudo-random state is seeded by a single integer as a parameter. Below shows some tests of the effect of the random seed on the score. Several experiments were run each with a different random state.

File Size	Undersample Size	F1 delta
10,000	2,050	3.4%
20,000	4,350	1.8%
500,000	51,435	.5%

These deltas are substantial. The Random State of the machine has a larger impact on performance than all our custom feature extractors combined. However, this problem does decrease with larger dataset sizes.

Further Work

Much can be said about improvements to the model we implemented, but our discussion is limited to the following four points:

- Better sampling across the data – our experiment used nearly 5% of the available Video Games Reviews dataset. This subset was taken form the most recent 100,000 reviews, representing only a few days in August 2015. It is possible that our model would see an improvement by training on reviews distributed across a longer period of time.
- Sarcasm Detection – Although the bag of words model accounted for most of the performance of our model, it is not sufficient to accurately decipher sarcastic sentiments. Research on this topic could help reduce the number of misclassified sentiments.
- User Updates – Amazon users are allowed to update their reviews days or months later. Typically they will add a new paragraph often prefaced with "Update: " describing their experience since the initial review. Users are also allowed to change the score of the review. This has the effect of a review which contains both language from a previous sentiment, along with more language of a different sentiment corresponding to the new score. For example, a user may leave a favorable 5-star review, then update the review

describing their frustration with the product breaking a month later. These reviews may cause confusion for the classifier which recognizes sentiments belonging to two classes. Our model could potentially detect an update paragraph, and omit the initial sentiment before the update, since it no longer represents the label for that review.

- Analyzing missed tuples – Within the scope of this project, we did not get the chance to manually review the tuples of the data which were mis-classified. Time spent analyzing these missed tuples could have revealed information about the ways in which our model falls short.

Conclusion

This project successfully implements a supervised machine learning model to predict star-ratings of Amazon Product Reviews. Our model managed an average f1 score above 67% using the Support Vector Classifier. The same model was able to perform binary classification with average f1 score above 93%. This score matches the results from previous research, which omitted three-star sentiment classification from its model.