Estimating Quantitative Customer Satisfaction From Written Reviews

Team: Hugo G. (CIS 5190), Daniel S. P. (CIS 4190), Michael R. (CIS 4190)

1. Abstract

Assessing customer feedback is vital for online businesses, though this can be difficult at scale when reviews are not accompanied by an easily quantifiable rating of customer satisfaction. In our project, we explore methods of estimating customer satisfaction (on 1-5 star scale) using written reviews and data about the product in question. Our project specifically focuses on Amazon products; we develop and analyze two separate algorithms to predict star ratings left by reviewers of Amazon products: the first (referred to as the 'stacked model') uses various out-of-the-box sentiment analyzers, product metadata, and engineered features as inputs in a three layered model. The second is our own sentiment model trained on Amazon review data ('custom sentiment model'). We compare the performance of our two approaches against a baseline set by XGBoost applied to TextBlob, one of the pre-trained sentiment analyzers we build on. Our project also aimed for a dataset contribution through use of feature engineering, though we found that model performance was not increased by additional product-related metadata. We find that our stacked model cannot compensate for the ineffectiveness of the pre-trained sentiment models relative to our custom sentiment model. That is, our custom sentiment model achieved the highest (balanced) accuracy and lowest MSE of 0.483 and 1.260, respectively. We believe that our findings have applications in the ecommerce industry, as our model could likely be applied to gauge customer satisfaction from 'informal reviews' (i.e. ones without quantifiable scores) from places like forums or Twitter, with reasonable accuracy.

2. Introduction

We are using sentiment analysis on Amazon product reviews in combination with information on the corresponding product in order to predict the overall star rating a reviewer gave. We use a dataset of 233 million Amazon reviews from 2018 spanning all product categories and a separate dataset on Amazon product information from 2018 (Jianmo et al.). Crucially, both sets contain a 'product id' variable which we use to join them. Due to the size of the dataset (and the cost we would incur buying gpu time), we train and test our model using Amazon reviews for the product category "Automotive", using ~600k of the ~1.7mm reviews. Note that the schema is identical across categories and our model is able to calculate category-average-related features when the dataset contains multiple categories, so our model would – with sufficient compute – theoretically run on the large dataset. As a performance measure we use the mean squared error as a loss function to calculate the error between the predicted ratings and the actual ratings of a set of Amazon reviews, as well as the overall accuracy. Note that the star ratings are represented as numerical grades (1 through 5) in this data.

Our first model – described further in the following sections – uses three different sentiment models on a review body and headline, together with other engineered features, followed by a layer with 5 different models, with an additional two layers of models stacked on top. Our second model is a CNN-based sentiment model that we trained on the Amazon reviews.

The baseline model that we are trying to improve on consists of a single XGBoosted (TextBlob) sentiment model directly predicting star ratings based on the content of the review alone.

Motivation

While users on Amazon must attach a star rating to their product reviews, we are interested in developing this model due to its applications elsewhere. On informal forums and message boards related

to products, users are ‘reviewing’ products yet often don't give explicit ratings (at least in an easily parsable format). From a retailer's perspective, knowing what quantitative rating a customer would give a product based on text they wrote is useful information. Customers who are commenting positive ‘reviews’ in a comment section could be automatically identified and prompted to leave an official review, etc. We are broadly interested in ecommerce and would be interested in deploying such a tool (provided that it is reasonably accurate).

3. Background

Paper A uses a modified Naive Bayes Model to predict sentiment of IMDB movie reviews. In order to further improve performance negation handling, n-grams, and feature selection were applied. The proposed model classifies reviews as either ‘positive’ or ‘negative’, whereas we aim to distinguish 5 separate rating categories with a different model & application. Paper B provides an analysis of an Amazon reviews dataset and studies sentiment classification with different machine learning approaches. However, it uses sentiment analysis to classify sentiment as ‘positive’, ‘negative’, or ‘neutral’, while we are trying to determine individual customers’ opinions on a product on the full 1-5 star scale and consider additional features.

A. Narayanan, Vivek; Arora, Ishan; Bhatia, Arjun (2013): Fast and Accurate Sentiment Classification Using an Enhanced Naive Bayes Model. https://paperswithcode.com/paper/fast-and-accurate-sentiment-classification
B. S. M. AlQahtani, Arwa (2021): Product Sentiment Analysis for Amazon Reviews. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwirjO_059n6AhUBElkFHTVsAzIQFnoECCUQAQ&url=http%3A%2F%2Faircconline.com%2Fijcsit%2FV13N3%2F13321ijcsit02.pdf&usg=AOvVaw0QBpdlSKa4oMqq3FexaXsD

In both cases, we aim for a more granular classification of the data, attempting to train our model with individual reviews and for every review classify over multiple categories. In our approach we explore this setup, and combine other features in the data to improve predictions.

4. Summary of Our Contributions

Algorithm: We have expanded on the work done in both papers by using alternate methods of sentiment analysis and stacking models. We use several pre-trained sentiment analyzers and do our own preprocessing. We feed features into a custom model pipeline, which combines sentiment predictions to predict the star ratings with a higher accuracy.

Algorithm: We separately made our own sentiment analyzer (CNN) and trained it on Amazon reviews to compare our two approaches. With this we achieved a higher accuracy than the baseline sentiment analysis, allowing us to detect unique patterns within the reviews that had an influence over the ratings.

Application/Dataset: We are using additional data compared to the prior work on Amazon reviews that we build on (AlQahtani). Specifically, we join product metadata with the review data as well as engineer additional features as described in the following section.

5. Detailed Description of Contributions

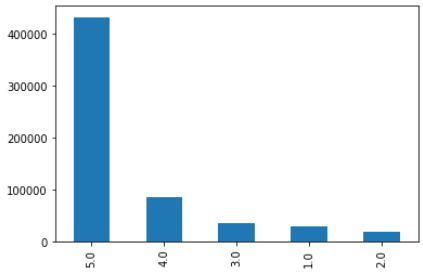
5.1 Methods

5.1.1 Preprocessing for Stacked Model:

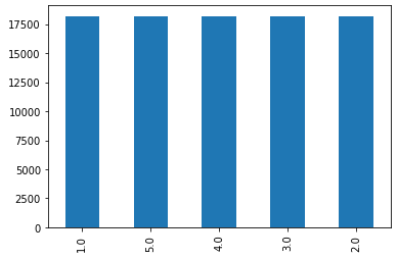
Balancing

The review dataset that we are using is highly unbalanced, with vastly more positive reviews than negative. We start by balancing the number of reviews such that each star rating is represented equally in all training and test data.

Initial Distribution of Data



Final Distribution of Data



Pre-Split Featurizers

After joining the review dataset with the product dataset, we make the following features: length of product title, sum of length of product descriptions, average sentiment of product descriptions, sentiment of product title, length of also_buy, length of also_view, size of review text, and number of contrast words.

We then apply our three 'out-of-the-box' sentiment analyzers to both the review body and the review title of each review. These sentiment models are Vader, TextBlob, and NLTK (all rule-based methods). The results of these predictions become features.

Splitting and Category Mean Features

Within the product metadata there are categories and brands which range in length from several main categories, to hundreds of brands, to thousands of sub-categories. In order to incorporate this data into the model, we do an initial run over the dataset to find the average rating for every category, brand and sub-category. This information will be part of the training of our model; when a prediction is made for a review, these features in the product metadata will be searched for in the lists of averages obtained during training.

In order to ensure a correct train-test separation, we have to split our dataset into train and test before calculating the category mean features and running the stacked model such that the category mean values do not encapsulate any data that will ever be used to test any layer of the model. Over this initial split, we obtain values for the average rating of categories, brands, and subcategories on the training data and extrapolate these values to the test data.

5.1.2 Stacked Model Design

Layer 1

Our first layer consists of six models in parallel: Linear Regression, KNeighborsClassifier (n=5), Random Forest Regressor (n=20), Adaboost Classifier (n=50), and XGB Classifier. We chose this many models partially as a learning exercise; Running them in parallel allowed us to easily tell which are well-suited to

the data (see appendix for the results). As shown in the diagram in the appendix, we are training Layer 1 on data that has no overlap with the test data for this layer nor the training/testing data for any other layer. As discussed in the preprocessing section we took care that the engineered features would not encapsulate information from other train/test sets.

Layer 2

We then applied an additional KNeighbors once we normalized the data we got from the first layer prediction. We chose specifically KNeighbors as we believed it was well suited to select the best model predictions from layer one.

Layer 3

We have two separate versions of layer three that we exchange as an experiment (discussed in the following section). The first is XGB, the second is a CNN. We optimized the CNN parameters.

5.1.3 Preprocessing for Custom Sentiment Model

In order to maximize training points for accuracy, our custom sentiment model is trained over train data 1 and 2 (see the appendix). Review text for this data is tokenized and limited to 100 words in length, with reviews lower than 100 being padded to 100 length in the embedding. Increasing this word count adds to much padding to distinguish features in the low-level reviews.

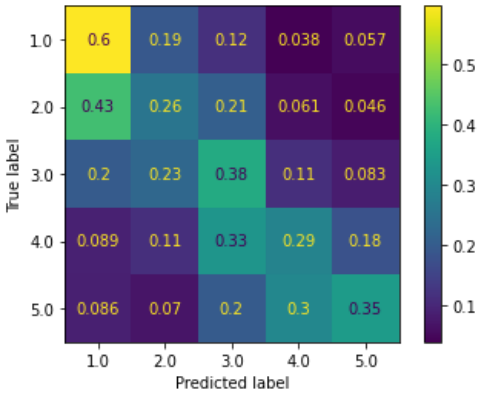
The tokenized text is then converted to a vector representation using GloVe pre-trained word embeddings (*glove.6B.50d.txt)*.

5.1.4 Custom Sentiment Model Design

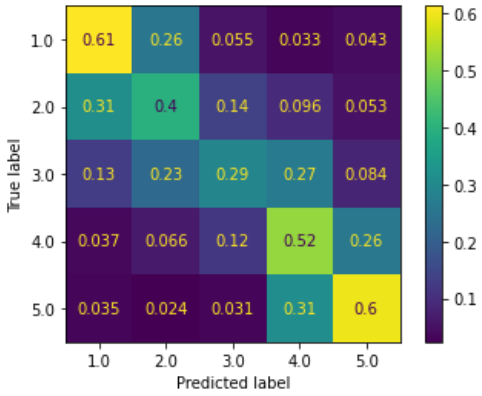
The CNN takes in the word embeddings and runs an initial convolution layer with 128 output filters, a kernel size of 3 and relu activation, into a global max pooling, into a linear layer with 10 nodes, and finally a linear layer with 6 nodes for the output dimensions.

5.2 Experiments and Results

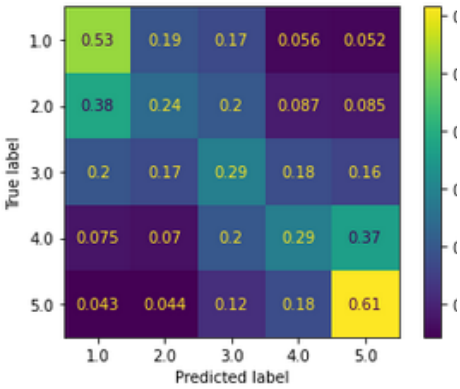
5.2.1 Overall Results



Stacked Model
Accuracy: 0.374
Mean Squared Error: 1.894
Root MSE: 1.365
Standard deviation: 1.348

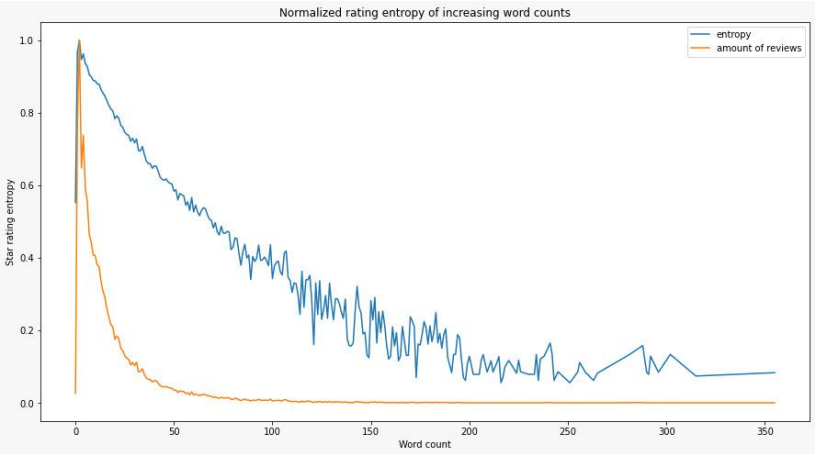


Custom Sentiment Model
Accuracy: 0.483
Mean Squared Error: 1.260
Root MSE: 1.1226
Standard deviation: 1.122



XGBoost on TextBlob sentiment prediction (baseline)
Accuracy: 0.311
Mean Squared Error: 2.69
Root MSE: 1.64
Standard deviation: 1.608

Both of our models outperform our baseline by a significant margin, both in terms of overall accuracy as well as mean squared error. Note that our models are especially improving in predicting star ratings between 2 and 4. We hypothesized early in this project that the difference between ‘1’ and a ‘2’ (or ‘4’ and ‘5’) would be a hard distinction for an out-of-the-box sentiment model to make, as they are largely used for only binary classification tasks.



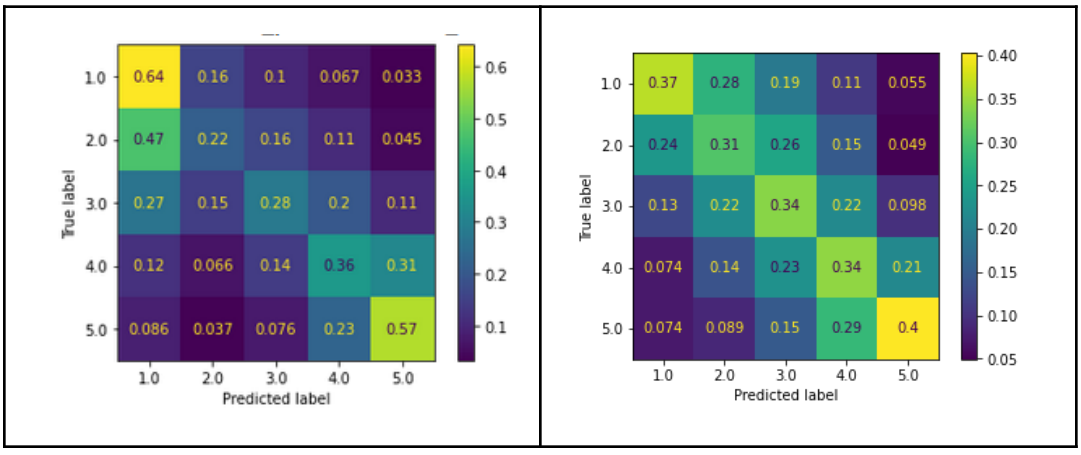
The differences between these models are emphasized by the size of reviews, reviews with a lower amount of words are harder to predict, and do not correlate with a certain rating. Our custom CNN model has a higher accuracy for these reviews as it can recognize small structures in the review text that do not necessarily denote sentiment, but are meaningful in the context of a review (i.e. ‘return’, ‘color’)

5.2.2 Experiments

The main question of our project was whether pre-trained sentiment models could together match or beat an Amazon-review-trained sentiment model, which as shown above was not the case. We were pleased that both nevertheless handily beat the single-sentiment-model benchmark. We performed several sub-experiments to analyze the approach we took on the stacked model.

Is the metadata even helpful?

We initially believed that product metadata could improve accuracy and reduce MSE as we hypothesized that some categories tended to have harsher language in their reviews, etc. We were surprised to find that this is not the case, with the simpler, no-product-metadata model outperforming as shown below. Early hints that this would be the case are in the correlation matrix (see appendix). However, we must note that the metadata vastly improved accuracy in the 2-4 star categories. The model without metadata heavily favors 1 and 5, which may not be optimal depending on the applications of the model.



[NO METADATA] XGB on KNN on Layer One *Accuracy: 0.41508* *Mean squared error: 1.80291* *Standard deviation: 1.3364*	**[WITH METADATA] XGB on KNN on Layer One** *Accuracy: 0.35117* *Mean squared error: 2.05139* *Standard deviation:* 1.43224

Note that the CNN – which we found resulted in superior performance compared to a layer three XGB – largely makes up for the worse performance from carrying metadata through the layers and using them to influence predictions. The KNN might very well have been a factor in the metadata reducing the results.

Deciding the Optimal Model for Layer Two

Initially, we used a K-Nearest-Neighbor classifier for layer two of the stacked model. This is also what we used for the final accuracy and mean squared error evaluation of our algorithm. As an experiment, however, we also used Random Forest as a layer two classifier, reducing the mean squared error of the layer two prediction by 0.06. At the same time, however, using Random Forest classification decreased the accuracy of the layer two prediction by 0.005. Therefore, we continued using K-Nearest-Neighbors classification. (See appendix for corresponding confusion matrices)

6. Compute/Other Resources Used

We used the GPU from Colab-Notebook on both CNNs. By doing so, we were able to build in more and larger layers as well as compute more data. This reduces the runtime of our CNN models from hours to minutes. Excluding the substantial time taken to load the datasets from the source, our model takes in the order of 30 minutes to run over 600,000 reviews, from generating initial features to final predictions on the test set.

7. Conclusions

Outcomes: Our challenges improving the accuracy of the stacked model showed us the importance of having models that are trained on specifically the type of data that you are using it on, as minor incongruencies can result in problems at scale. Specifically, the sentiment analyzers that we used out of the box being trained on movie review datasets in the case of TextBlob, makes them unsuited to the unique application of product reviews, as is shown by our custom model vastly outperforming the three rules-based out-of-the-box analyzers. We believe that our custom sentiment model might have useful applications applied to scan ‘informal’ reviews on sites like reddit and twitter. Official review sites struggle with spam, and perhaps a tool that can summarize written reviews could more accurately encapsulate customer feedback.

For the future: Some of the features we have generated in our model require an initial parsing of the amazon database in order to use product reviews from different columns. When applying this model to a different database (one without initial ratings data), if one of these feature values is missing (for example, a brand that does not appear in the amazon database), our model would not be able to use that value. It would be possible to add an initial stack to the model predictors that would attempt to predict these missing values (using existing values with product and comment metadata), we predict this would increase the model’s ‘portability’ and make these features more robust.

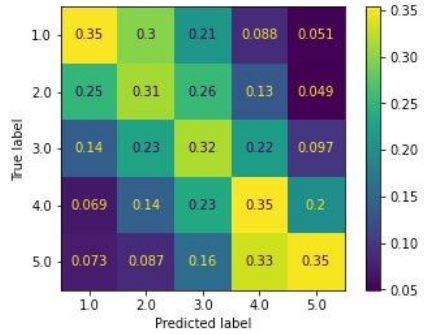
Ethical and environmental considerations: The model has the potential to generate quantitative metrics for companies that have a dataset of qualitative reviews, in order to provide a more analytical measure of consumer satisfaction for certain products. By having a more complex stacked design, it allows for use on data with different features without retraining some of the model sections, saving computation time and resources.

(Exempted from page limit) Supplementary Materials (Appendix)

Appendix 1: Layer 2 Experiment Correlation Matrices

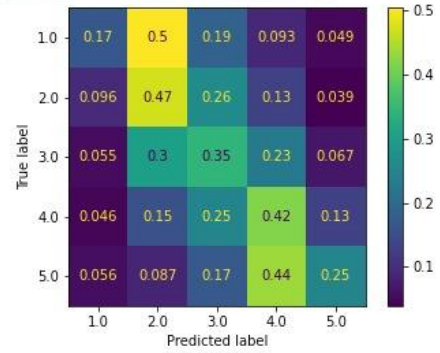
KNN Layer Two:

```
accuracy score:  0.33633846809114787
mse: 2.022361042095549
root mse: 1.4220974094961107
```

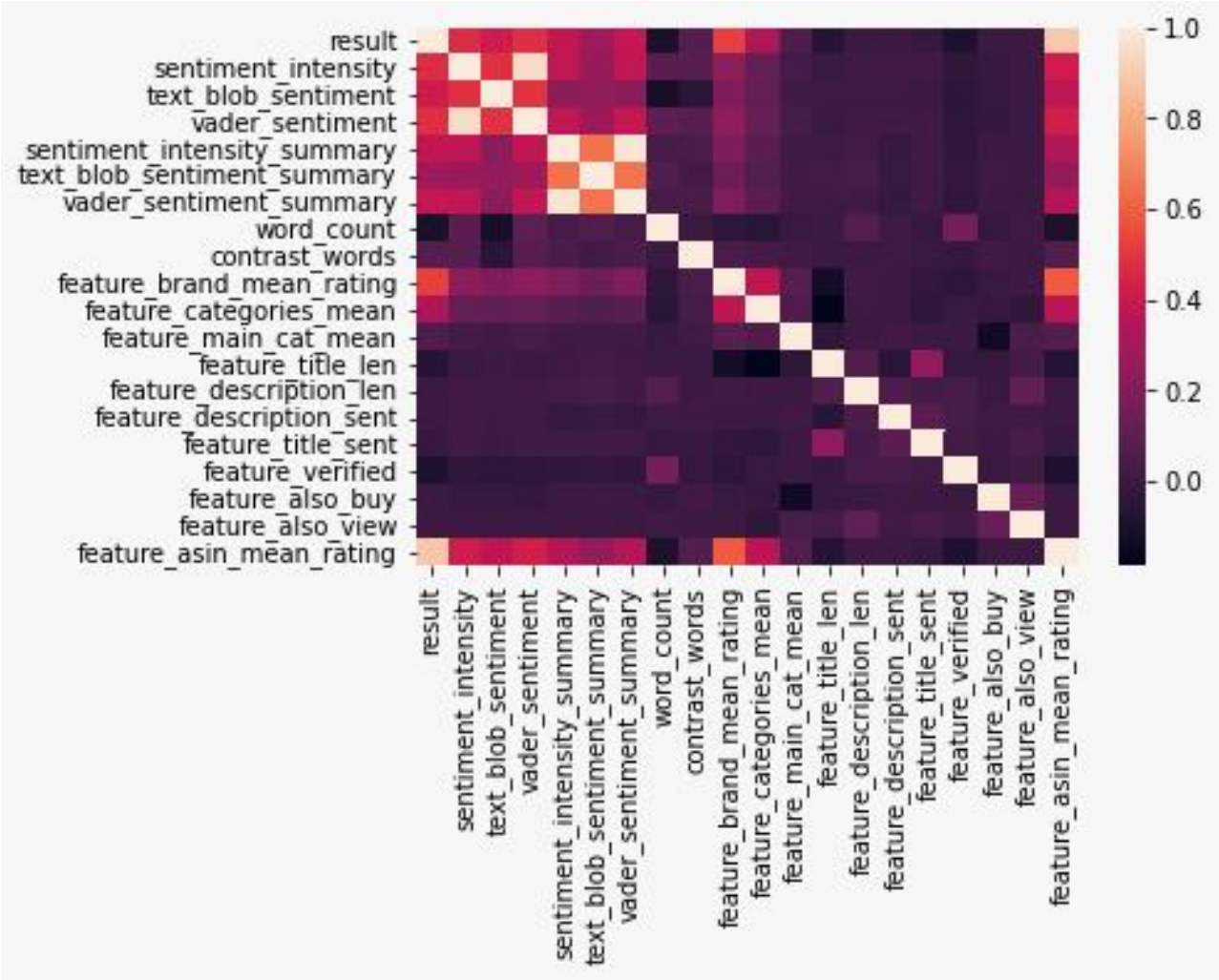


Random Forest Layer Two:

```
accuracy score:  0.3317952722368141
mse: 1.8531979839568395
root mse: 1.3613221455470559
```



Appendix 2: Correlation Matrix



Appendix 3: Stacked Model