Michael Nguyen

How to spot helpful reviews?

Introduction

Customer review is one of the most critical factors for a successful online business. Based on data from FanandFuel in 2016, more than 90% of customers read reviews before making a purchase. In addition, research from Northwestern University indicates that people tend to buy a product with reviews 270% more than those without. With the importance of reviews, it is thus necessary to weed out the fake ones. Fake reviews hurt not only customers but also sellers by destroying their reputation and trust. Therefore, it is incredibly vital for businesses to develop a model to classify whether a review is fake or not. Unfortunately, it's not easy to obtain a dataset with reviews that have already been classified as real or fake. In addition, not all honest reviews will help customers make their decisions. Perhaps instead of finding whether a review is genuine or not, one should discover whether a review is helpful or not. In this project, we will develop a model to spot out whether a review is helpful or not based on users' feedbacks. The codes performed in this project can be found at: mnguyen494/Capstone-SpringBoard-2: Project of classifying Amazon reviews. (github.com)

Dataset

In this project, we will use a dataset provided by Amazon that can be downloaded on tensorflow.org. Each row in the dataset corresponds to an individual review with the following features:

- customer_id: the unique id of the customer
- market_place: 2 letter country code of the marketplace where the review was written
- review_id: the unique id of the review
- product_id: the unique id of the product
- product_parent: the unique id of the family which the many products belong to (for example: a red pen and a blue pen belong to the same family)
- product_title: the title of the product
- review_headline: the headline of the review
- review_body: the body text of the review

- review_id: the unique id of the review
- review_date: the date of the review
- star_rating: the rating of the review that ranges from 1 to 5
- total_votes: the total votes on the review
- helpful_votes: the number of helpful votes on the review
- verified_purchase: a marker that verify whether a reviewer actually purchase the product
- vine: a marker that indicate whether a reviewer in the VINE program

In total, this dataset contains over 130+ million customer reviews on Amazon in TSV files. This number of reviews is way too big for a local computer to handle. Thus, we will limit our analysis to reviews of office products. With this, our data contains about 2 million customer reviews.

Data Cleaning

Upon downloading the dataset, we discover that the dataset needs serious cleaning. It contains 22 columns, not 15 columns as indicated in the above description. The extra columns have mostly null values, so we decided to drop them. We then drop all rows that contain any null value because its amount is relatively small compared to the entire dataset. Next, we pay attention to the remaining columns. Many columns such as "star_rating" have mixed types consisting of string, float, and int. We then make sure that the types of all columns are consistent and appropriate. At the end of the cleaning, our dataset has over 2 million rows and 15 columns.

Data Wrangling

Before further analysis, we need to decide on a metric to evaluate how helpful a review is. One apparent feature is the "helpful_votes." However, as shown in Fig. 1 (left), the number of helpful votes for a review can vary widely, from 0 to 40,000, with many outliers. This will make analysis across the reviews difficult. Thus, we decide to create a new feature:

helpful_votes_ratio = helpful_votes / total_votes

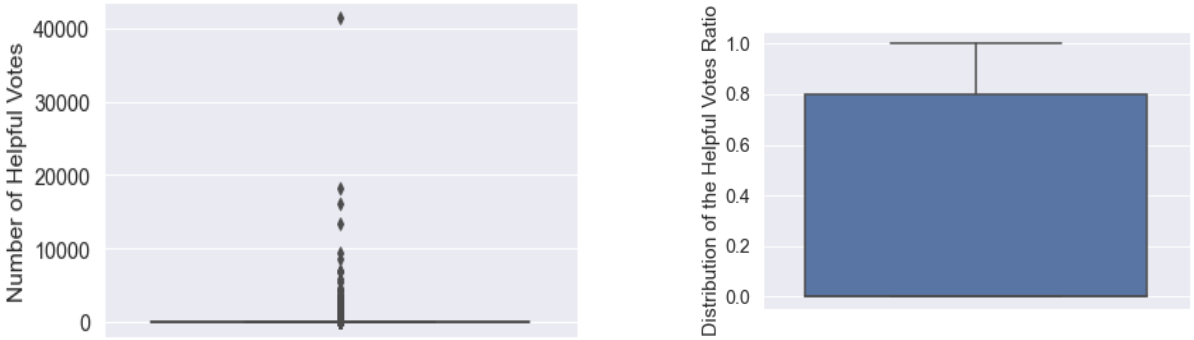

Fig. 1: (Left) The distribution of the "helpful_votes" for all the reviews.
(Right) The distribution of the "helpful_votes_ratio" for all the reviews.

With this new feature, the most helpful and trustworthy review will have a value close to 1, while the bad review will have a value close to 0. As shown in Fig. 1 (right), the range of the new feature is much more standardized.

Next, we look at some of the outliers. Brief analysis indicates that the reviews without verified purchase have a huge number of outliers. In addition, the number of not verified purchase reviews is relatively tiny compared to the rest of the data. For these reasons, we decide to drop all reviews without verified purchase. The final dataset has about 2.2 million reviews with 14 features. (we have taken out the "verified_purchase" feature).

Exploratory Data Analysis

Now, let's take a closer look at some of the features. The most important feature is the "helpful_votes_ratio" because we will use it as a metric for modeling. Below, we plot the histogram of the reviews according to their "helpful_votes_ratio." The graph suggests that most of the votes are either very useful or useless which little in between.

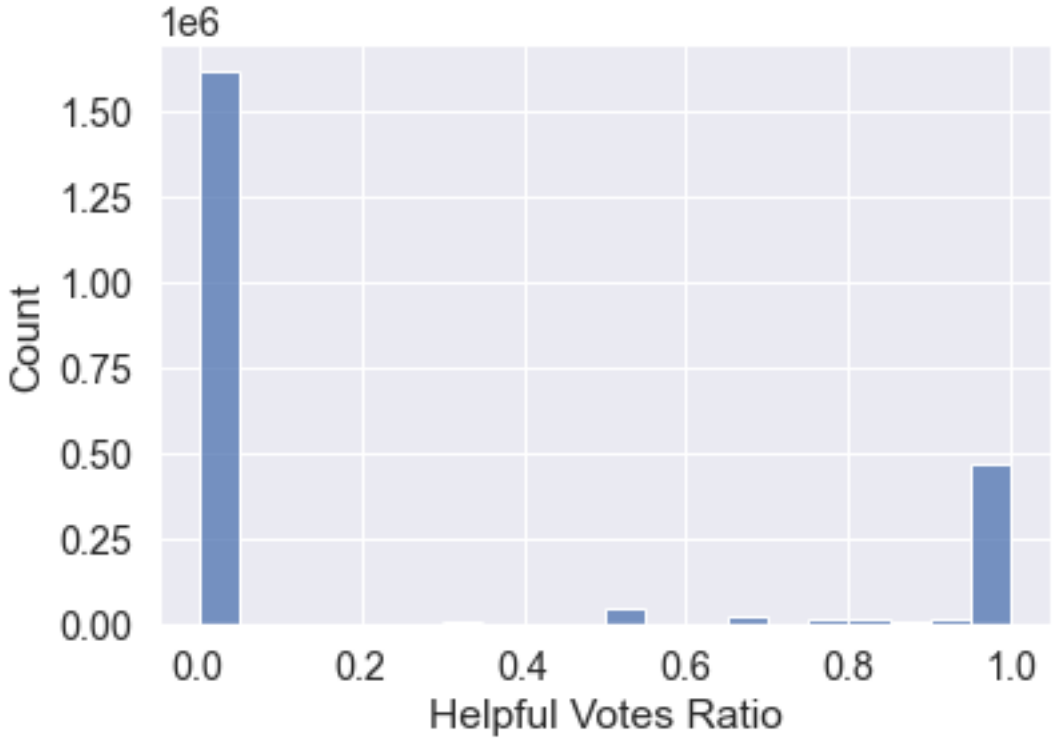

Fig. 2: Histogram of the reviews according to their "helpful_votes_ratio". The figure shows the reviews are very bipolar.

The other features that can be helpful for our analysis are: "product_id," "product_title," "product_parent," and "customer_id." Most of the values in these features are string id that does not hold any particular meaning except for identification. To remedy this, we will number these values according to their appearance on the dataset. Thus, customers with many reviews will have a high number, while customers with few reviews will have a low number. After transforming these features, we will add the term "arrange" to their column names to distinguish them from the old features. (Ex: "arrange_customer_id" from "customer_id.") This way, we can consider the effect of how often a product is reviewed, how often a customer reviews a product, etc. Another essential feature that is not originally in the dataset is the length of the review. We will create another column, "review_length," to indicate how long a review is, based on its characters in the dataset.

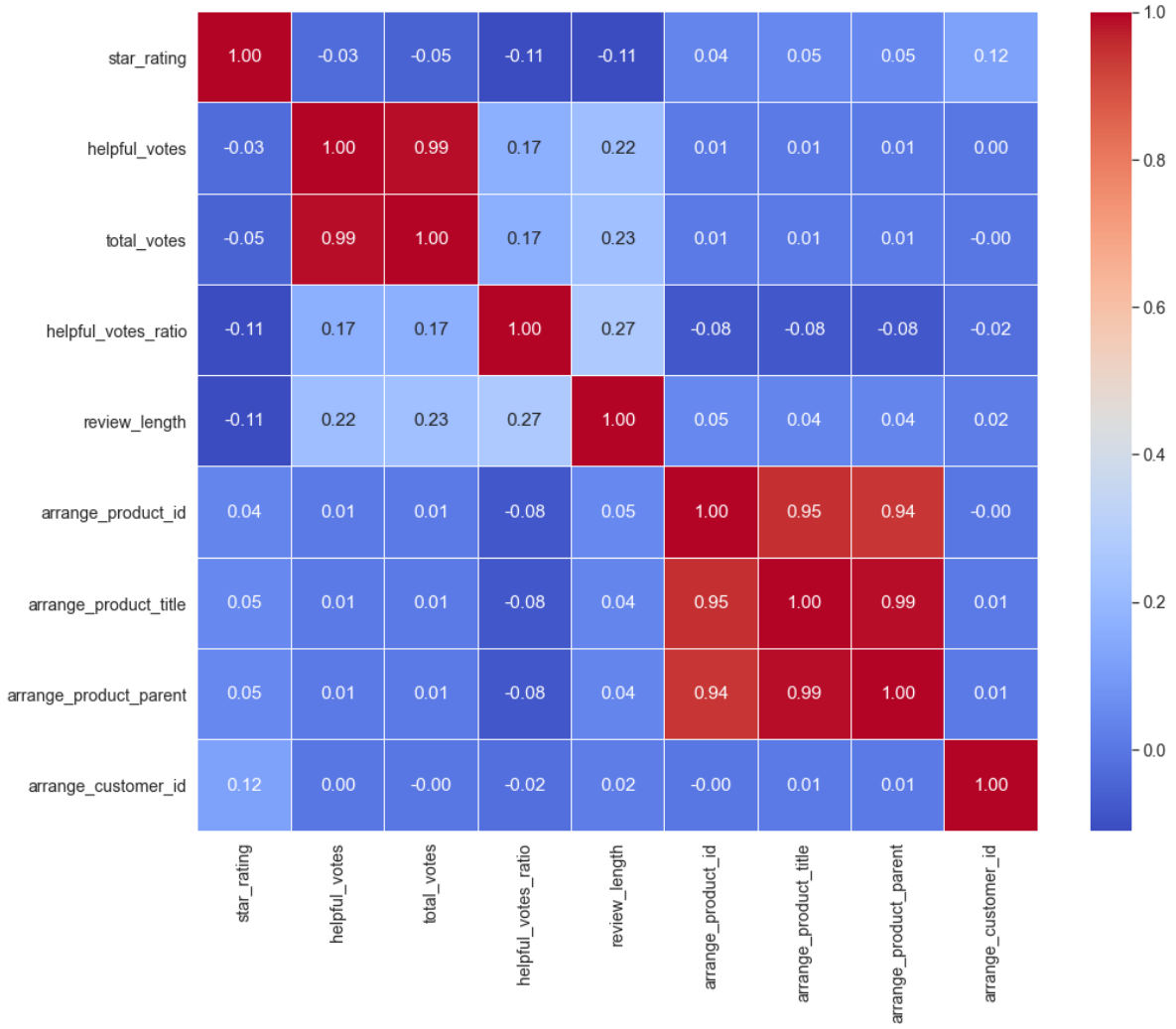

Fig. 3: Correlation matrix of many important features in the dataset.

Now, let's check how the features in the dataset correlate with each other. In Fig. 3, we have plotted the correlation matrix of many essential features and the metric, "helpful_votes_ratio."

According to the correlation matrix in Fig. 3, there's little correlation for the "helpful_votes_ratio" to features such as the products and the identity of the customers giving it. The "helpful_votes_ratio" does correlate strongest with the "review_length." Thus, we take a closer look at this relationship by plotting the "helpful_votes_ratio" vs. "review_length" below. The plot shows that longer reviews tend to be more helpful than shorter ones. This makes sense since short reviews such as: "It's a good product" or "I like it" do not contain much information and are not helpful to other customers.

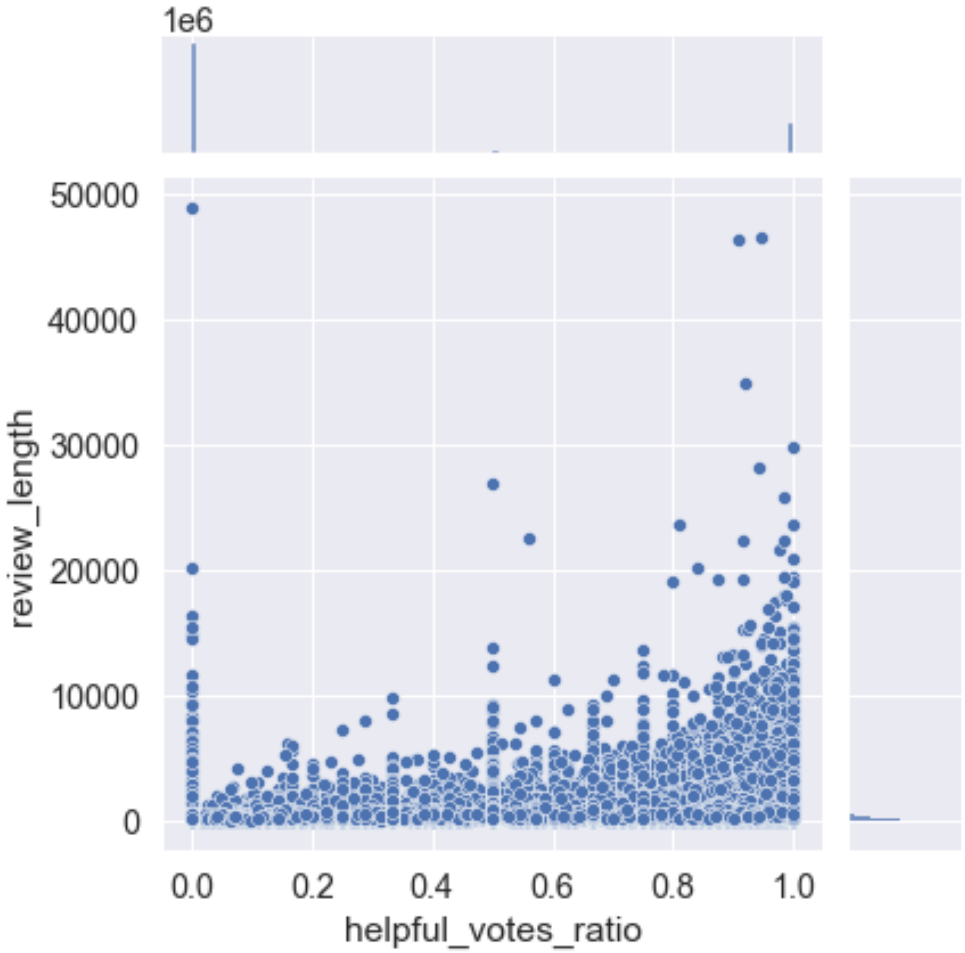

Fig. 4: “helpful_votes_ratio” vs “review_length”. The figure indicates that longer reviews tend to be more helpful than shorter ones.

Michael Nguyen

Reviews Cleaning

In our previous section, we have looked at external features without looking at the actual review itself. We will now prepare our reviews for modeling. Throughout this process, we will use the NTLK library with its corpus extensively.

First, we will tokenize the reviews. Tokenization is the procedure of extracting words from a sentence. For example, the sentence “I am doing an NLP project.” can be tokenized into “I,” “am,” “doing” “a,” “NLP,” “project.” In this example, we have extracted one token at a time. These tokens are called unigrams. Alternatively, we can extract two or three tokens at a time. Such tokens are called bigram and trigram alternatively. And in general, depending on the project, one can extract n-grams tokens. For this project, we will consider unigram and bigram tokens.

The next step for our cleaning is removing stopwords. Stopwords are words that appear too common in most sentences. For example: “the,” “is,” “are” are common stopwords. Their purpose is mainly to help the construction of sentences and do not contribute to their meanings. Despite their necessity, their frequent appearance will hinder our ability to extract essential features from the sentences. Thus, they need to be taken out. For this project, we will use NTLK’s stopword corpus to remove the stopwords in our sentences.

After removing the stopwords, we will lemmatize the reviews. Lemmatization is the process of returning the base or dictionary form of a word known as lemma. For example, the term “kept” and “keeps” can be lemmatized into “keep.” This process will keep our corpus concise and easier for feature extraction.

Reviews Vectorization

After all the cleaning, now we can start to vectorize the texts to feed to our models. One popular method to do this is the Bag-of-Words (BoW) model. In this method, all our reviews will be represented by a giant matrix that stores the number in which a token appears in a review. Each row in this matrix corresponds to a review, and each column corresponds to a token that appears in the entire text.

Despite its popularity and simplicity, the BoW model has many serious drawbacks. One drawback is that it will heavily emphasize tokens that appear too often across all the reviews. For example, a review about a pen will have many words such as “pen” and “ink.” These common terms might not convey as much information as the rarer terms do. To avoid this drawback, we will utilize a method called Term Frequency – Inverted Document Frequency or TF-IDF.

Similar to the BoW model, all the reviews will be represented by a matrix. The value in the matrix, however, is no longer the appearance of the token in a review but a TF-IDF value calculated according to the formula:

$$\text{TF-IDF} = \text{TF}\log_{10}(N/\text{DF})$$

Here TF of a token is the frequency it appears in a review. N is the number of reviews, and DF is the number of reviews with the token under consideration. With this, too common tokens across all the reviews will not contribute too much to our models. With this, we are ready to combine the TF-IDF matrix with important external features that we prepared earlier: "review_length," "star_rating," "arrange_product_id," "arrange_product_title," "arrange_product_parent" and "arrange_customer_id."

Pre-processing and Training Data Development

Before we do the model, let us look at the distribution of the "helpful_votes_ratio" in Fig. 2. The bipolar of the distribution suggests that we should model using classification instead of regression. We then create a new column called "classified_review." In this column, reviews with "helpful_vote_ratio" bigger than 0.5 will be assigned the value one while the rest will be given zero.

The distribution of "helpful_votes_ratio" also reveals that helpful reviews are much smaller than useless ones. Because of this skewness, a model might give very good overall accuracy for identifying the bad reviews but does a terrible job in identifying good reviews. One solution to remedy this skewness is undersampling the bad reviews in our training set. With this, both good and bad reviews will be equally represented in the models, as shown in Fig. 5. With undersampling, the accuracy of the model might

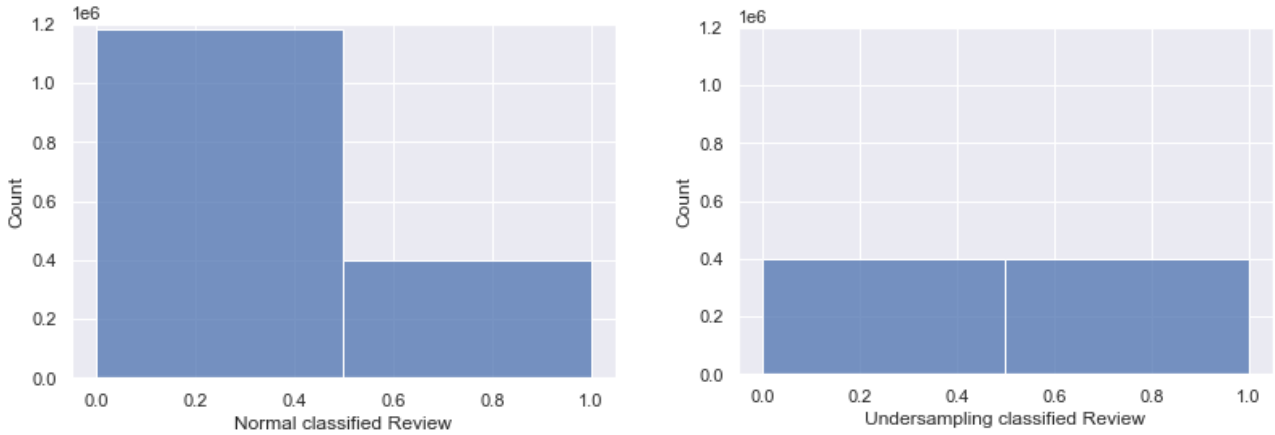

Fig. 5: (Left) The normal distribution of the "classified_review" for our training set. (Right) The undersampling distribution of the "classified_review" for our training set.

decrease, but the f1 score will increase. Unfortunately, without a better understanding of how reviews are connected to a more practical metric such as sales, it is not clear whether the undersampling training set is preferred to the normal one. We will thus make models for both the undersampling training set and the normal training set and discuss their pros and cons.

Model Evaluation

We now have several parameters in choosing our models:

1. We have two kinds of tokens: unigram and bigram.
2. We have two training sets: an original one and an undersampled one.
3. We will choose five algorithms for our training: logistic regression, random forest, gradient boost, naïve Bayes, and linear discriminant analysis.

In Fig. 6, we plot all the accuracies of all the models. The blue bar represents models with unigram tokens and undersampling. The red bar represents models with bigram tokens and undersampling. Next, the yellow bar represents models with unigram tokens without undersampling. Finally, the green bar represents the model with bigram tokens

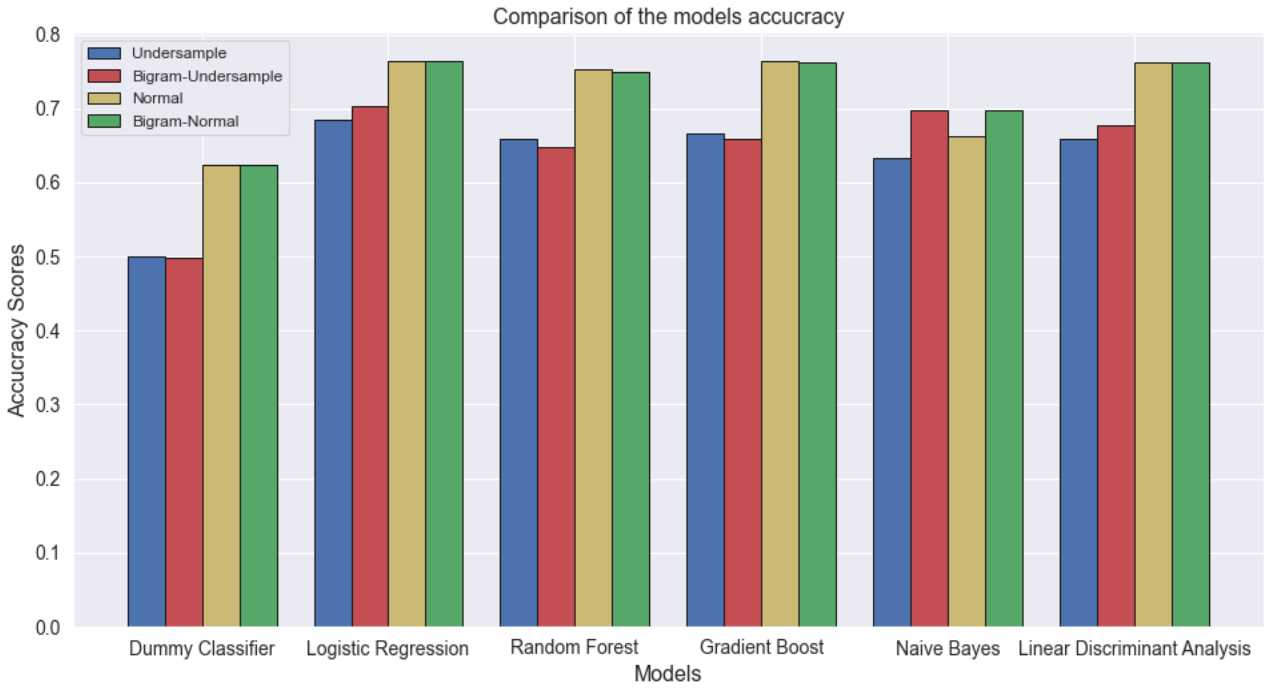

Fig. 6: Accuracies of various models. From left to right: dummy classifier, logistic regression, random forest, gradient boost, naïve Bayes, and linear discriminant analysis. Blue bar: models with unigram tokens and undersampling. Red bar: models with bigram tokens and undersampling. Yellow bar: model with unigram tokens without undersampling. Green bar: Model with bigram tokens without undersampling.

without undersampling. As we go from left to right, the algorithms used are dummy classifier (a dummy model that used the statistics of the training set to classify the review randomly), logistic regression, random forest, gradient boost, naïve Bayes, and linear discriminant analysis. In term of accuracy, the models without the undersampling outperform the ones with undersampling. This is to be expected for data with skewness. Whether the tokens are unigram or bigram does not seem to contribute to the accuracy. To conclude, if accuracy is used as the primary metric for evaluation, then gradient boost with unigram tokens and without undersampling is the best model.

In Fig. 7, we compare how the models perform in term of the f1 score. Now the models with undersampling outperformed the ones without. Again, the bigram tokens do not improve the models. The best model in term of f1 score is gradient boost with unigram tokens and undersampling. Thus, it seems that gradient boost is the best algorithm for all situations.

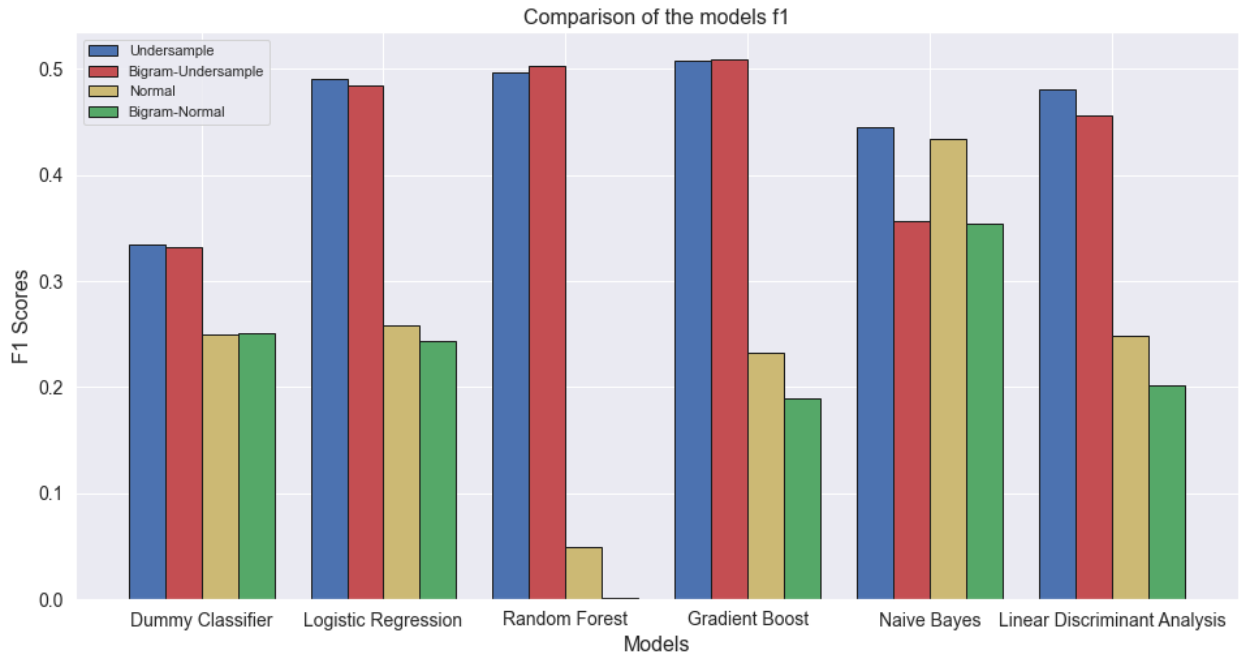

Fig. 7: F1 scores of various models. From left to right: dummy classifier, logistic regression, random forest, gradient boost, naïve Bayes, and linear discriminant analysis. Blue bar: models with unigram tokens and undersampling. Red bar: models with bigram tokens and undersampling. Yellow bar: model with unigram tokens without undersampling. Green bar: Model with bigram tokens without undersampling.

Michael Nguyen

Conclusion and Future Direction

In this project, we have created models to determine whether a review is helpful or not using TF-IDF and other external features. Unfortunately, without a good understanding of how reviews affect sales, choosing a model based on its accuracy or f1 score is not straightforward. Further analysis with marketing is required to make the decision. In further improving our model, we should add other essential features such as the prices of the products to our analysis. In addition, we can also implement more advanced models such as recurrent neural networks.