Part 1: Identifying a dataset and EDA

The data set we have decided to work with is the Amazon review dataset in 2018. Specifically we are looking at a subset of the data which follows a 5-cores model such that each of the remaining users and items have 5 reviews each. The specific genre of reviews we are using for the 5-cores amazon data is the prime pantry data which consists of 137,778 data records.

Analyzing the ['reviewerName'] column we find that there is only 12,318 unique users in the dataset. Versus 137,788 reviews. Analyzing the ['asin'] column, which houses the product id of the product being reviewed, we find that there is only 4,970 unique items in the dataset. Versus 12,318 unique users. Looking at our pandas data table above created from the amazon dataset we can see there are 12 different columns. We will be dropping columns ['vote','image','style'] due to all entries of those columns being NaN entires.

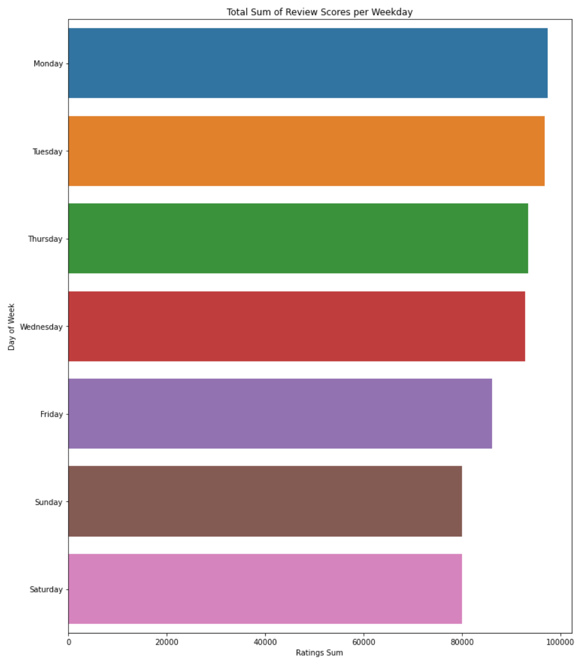
After dropping columns ['vote', 'image', 'style'] we now are left with 9 columns. There are two columns which determine the date/time when a review was written which we want to explore. ['reviewTime'] which represents the date as (d/mm/yy) and ['unixReviewTime'] which respresents the date in unix time - which is easier to interact and explore with in python.

One of the questions we are interested in exploring is how users rated products based on the specific day of the week. In order to analyze by the day of the week, we needed to create a new variable/column corresponding to which day of the week a review was written. We use the ['unixReviewTime'] column in order to extract the day of the week as a integer from unix time using the datetime package. We then convert the integer day of the week value to a string value ie. Sunday, Monday etc.

We then were able to create two new columns in our dataframe. ['dayOfWeek'] which represents the day of the week as a integer when a review was written. ['dayOfWeekName'] represents the day of the week as a string for when a review was written.

Using this new column, we wanted to explore which days had the highest number of reviews that were written and the days which had the highest sum of ratings.

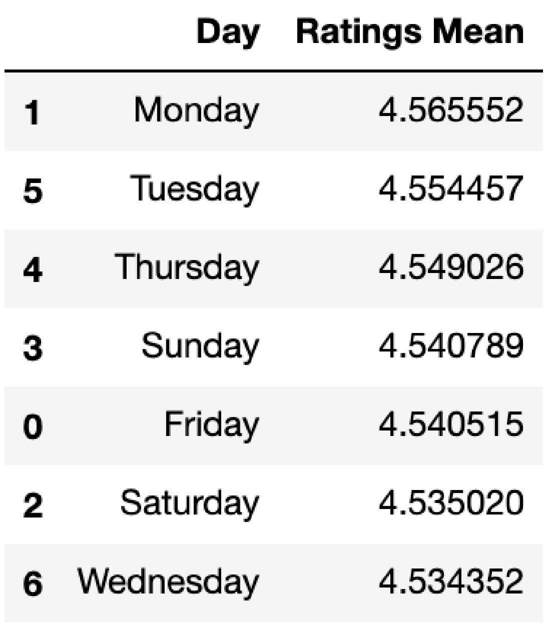
We can see from our analysis that Monday has the highest sum of ratings in 2018 with a total 9733. This is further represented in the graph below, with Tuesday being the second highest ratings sum.



While analyzing our graph and table we found an interesting trend in which the week days had a larger sum of ratings. In order to verify this statement we also graphed and counted the amounts of reviews per day of the week.

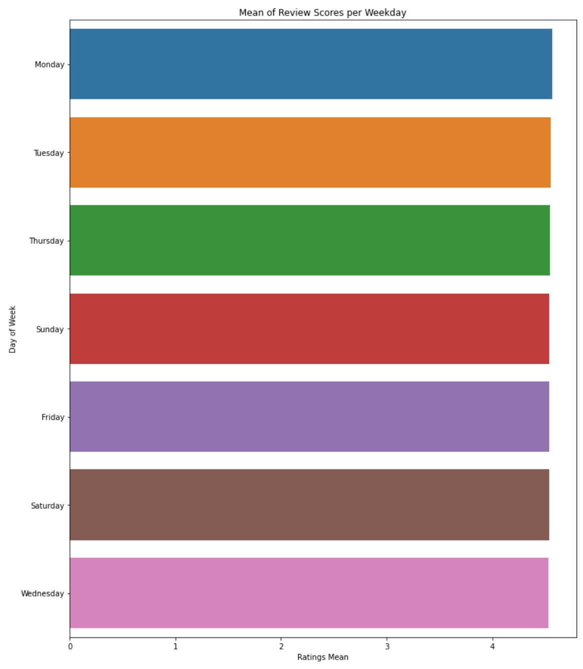
This ordered table showcasing the counts of reviews per day of the week interestingly has the same descending order as our sums of review ratings table and graph. This could suggest that Monday and Tuesday may not be the day with the highest ratings, due to the fact that Monday and Tuesday have the highest number of reviews out of all days of the week in 2018.

In order to corroborate this statement and analyze which day has the highest ratings, we decided to calculate the average rating from 1-5 on each day of the week in 2018. This is due to the reasoning that the averages mitigates the imbalance of number of reviews per day of the week, since Monday has over 2000+ more reviews that Saturday and Sunday.



	Day	Ratings Mean
1	Monday	4.565552
5	Tuesday	4.554457
4	Thursday	4.549026
3	Sunday	4.540789
0	Friday	4.540515
2	Saturday	4.535020
6	Wednesday	4.534352

We can see that Monday, Tuesday and Thursday are the first second, and third highest days of the week with rating averages in 2018. This follows the same ordering as our graph and table above exploring the sums of ratings. However, we do notice a deviation from the sums of ratings graph and table which state the weekends had the lowest
ratings in 6th and 7th standing, whereas the ratings mean graph and table showcase the weekends (Sunday and Saunday) had a 4th and 6th standing.



Part 2: Identify a predictive task on the dataset

Using our explanatory analysis of the dataset we want to try and predict the ratings of a product by a user. We are looking at ratings as a feature of a users review text.

In order to test our models with baselines, we will be dividing our data set into three sections. 80% of the dataset will be used for Training and Valid subsets. 20% of the dataset will be used for testing our model. We will be feeding our model the ['reviewText'] column and extracting a output of predicted ratings for each row in the column. To calculate the accuracy of the model we will be using the sklearn package and its f1score metric to calculate the predicted rating accuracy versus the actual rating.

One of the models that we could use from class is a jaccard simalarity based model which takes (user, product, rating) pairings and predicts the rating a user would give to the book. The jaccard portion of the model can be swapped out for different similarity based metrics, such as the cosine metric or pearson model.

However, based on our work in Homework 3 and further in Assignment 1, similarity based models reach a plateau in growth of accuracy. This is where we can transition over to bag of word feature vector models, that take advantage of the ['reviewText'] column and the reviews that users right.

Based off my analysis in Assignment 1, we want to take the bag of word feature vector models and implement a more complex model on it. This is in reference to https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/ report which I utilized in Assignment 1. This model makes use of Multi-Label Classification of users review text, analyzing the frequency of the 10,000 most common words in all reviews and their prevelance in each individual review. The reason we chose to use 10,000 words versus HW3's 1,000 words is that I observed an increase in accuracy of my model in Assignment 1 when increasing the number of most common words the model is being trained on to 10,000 words.

Part 3: Describe your model

Overall, we will be using the ['reviewText'] column (users reviews) to predict a users rating of a product. We want to implement this by firstly cleaning the 'reviewText' column from filler words and punctuation. Secondly, we want to find the 10,000 most common words across all reviews. Third, we then want to TF-IDF feature vectors counting the instances of those 10,000 words in each review. Lastly, we want to create a logistic regression model from the TF-IDF feature vectors in order to predict the ratings for each review.

We will split the data into a training set of the first 100,000 rows, and a testing set of the last 37,552 rows.

We then will remove all punctuation from the review text, placing the new cleaned review text in a new column called ['cleanedReviewText']. We then proceed to remove all filler and stop words from the cleaned review text. We make use of the nltk package stopwords. The reason we remove the filler words is to remove redundant variables that will skew the model when training. These words include general words such as "is, like" and more, which are relevant in most reviews.

Using the new cleaned review text we can begin dividing our overall training data into training and valid datasets to train our model. We are using sklearn's train_test_split model to divide into a 80/20 split of training and valid datasets. We can then create Tfidf vectorized models to fit and transform our Xtrain data.

Using out fitted model, we can create a logistic regression with a C coefficient of 2. Using the OneVsRestClassifier from the imported sklearn package we create a model that can be fitted to our Tfidf vectorized Xtrain models and ytrain (training ratings) data. Using this fitted model we can test on our xValid Model

Comparing our predicitions to the yValid subset we use f1_score from the sklearn package to calculate a accuracy score. Overall our model has a 0.76515 accuracy.

We repeat the same cleaning review texts for the test dataset 'dfTesting'.

We then repeat the same Tfidf vectorizers for the cleaned review text column. Using the vectorized column, we can use our model from training and predict our ratings. Using f1_score we get a accuracy of ~0.73748.

Part 4: Describe literature related to the problem you are studying

We obtained the data from https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews It follows a standard format that most reviews datasets typically follow. I.e., it has a unique product identifier, a unique user identifier, an overall rating, a rating text, etc. One we're familiar with that is similar to this one is the good reads book reviews dataset. Generally, there are many variations of Amazon reviews datasets (e.g., Amazon UK, Amazon shoes, etc), and they are all broadly used for educational and research purposes.

In the real world, these review datasets tend to get quite big. This is because marketplace platforms tend to be centralized and garner many users, which generates a large amount of reviews. These days, they can go anywhere from 30 GB to 1 TB. This means that an industry standard way of analyzing them requires scalable distributed systems such as Apache Spark, Dask, Modin, and more. In addition to scalable tools, certain best practices are followed to be able to handle such a large amount of reviews efficiently, such as extracting the desired rating from the helpful column and converting it from string to int, dropping stop words from review text, converting ints to np.int16 or in extreme cases np.int8 so as to create more memory storage space, dropping columns that aren't useful for your purposes, repartitioning the data, performing operations aggregately, varying the partitioning from row-wise to column-wise and vice versa depending on your desired result, and many more.

In our implementation, we used TF-IDF vectorizer, which ended up working pretty well. However, it is considered the "least good" of the commonly used models. In industry, more sophisticated models, particularly of the BERT family, are used, which actually perform the best out of TF-IDF, Doc2Vec, Word2Vec, and LSTM/GRU. On average, BERT models get a 10% increase in accuracy metrics. For us, we have ~74%, whereas BERT models tend to get in the ~82% range. This should be taken with a grain of salt, however, because this perdective task (predicting ratings from review text) isn't as useful as other tasks, such as predicting whether a user will buy another similar item, or predicting rating from previous purchases, etc.

Part 5: Describe your results and conclusions

Our model got a ~76.5% f1 score on validation and ~74% on test set. This is slightly better than what TF-IDF vectorizer models typically obtains for review-text to rating prediction models. Why our model performs better is likely a result of our data preprocessing and feature selection. We opted for "the simpler the better" route and chose not to include irrelevant features. Although, we definitely could have achieved better performance from engineering new features, or processing the review text in a more sophisticated NLP-native way. In industry, more sophisticated NLP-based pre-processing tends to produce better results than what we obtained, so that is certainly an option. From a pre-processing perspective, we chose to remove stop words, even though we don't have any memory issues, because we considered it noise that would hinder the model's ability to find signal, and we wanted to remove redundant variables that will skew the model when training. Additionally, we removed punctuation and non alphanumeric characters, as those also can be ruled out as noise.

Something that didn't work well was using the day of the week as a predictor variable. It turned out that the mean reviews for each day of the week were basically the same, and as expected, there was no correlation between day of the week and rating variability. One might expect, for example, that non-weekdays may have more generous ratings, but this was not the case. Thus, this approach was not used as it did not lead to useful results. And we could not find any usage of this approach in the literature.

Overall, our model wasn't industry-standard performance, but still did learn how to predict ratings a lot better than randomness (54% better!) so it's still considered useful and can be put to use. The main interpretation we have for why this is the case is that we have very high-leverage data, and we processed it and architected the model well. Meaning, the data we have contains the review text column which contains rich information that is a strong predictor of rating. However, to extract the information from the text, one must go about processing it properly and feeding it to a well-architected model. In terms of processing, we followed the best practices taught in class and used in industry, which helped eliminate redundancy and noise. As for the architecture, TF-IDF is a strong approach that has been broadly used in the literature and played a key role in our performance achievements.