Sentiment Analysis and Key Phrase Extraction Using Machine Learning

Final Report

AIDI 1002: AI Algorithms

Class Instructor: Marcos Bittencourt

Michael Molnar

Durham College # 100806823

Submission Date: December 17th, 2020

Executive Summary

Reading and analyzing customer comments, reviews, blog posts, and social media posts is a slow and labourious endeavour for a human being, though it is an essential way by which a business learns about itself and its customers. Machine Learning allows for the acceleration and automation of this process through text classification and sentiment analysis. A learning algorithm can be trained on pre-labelled data to discover associations between the text and the labels, and then is able predict labels for new and unseen text. This removes the need for predefined and human authored rules for differentiating between the class labels.

The scope of this project will be using Natural Language Processing to develop a Sentiment Analysis model that will predict whether a selection of text represents a positive, neutral, or negative feeling about a business and its offerings. It will then extend to extract the key words or phrases of the text.

The final deliverable is a fully functioning and stylish web application I have built using Flask. The app takes user text and makes a sentiment prediction on it using my model. The results are displayed, along with a bar graph displaying the probabilities that the model has determined for the sentiments.

Business Problem Statement

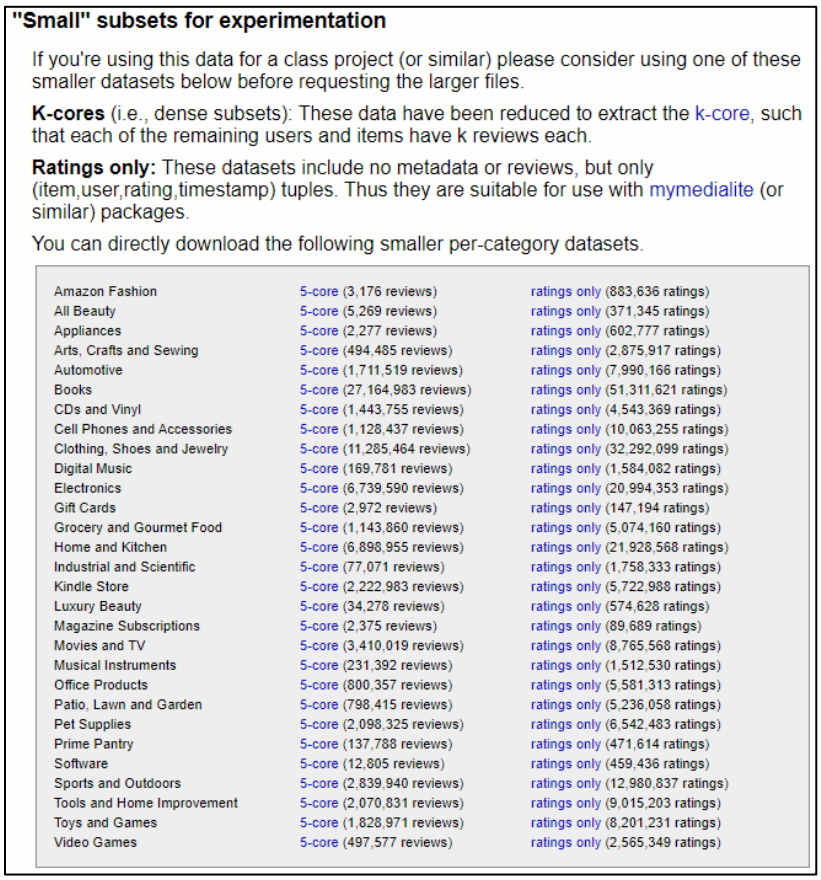
It has been estimated that 80-90% of all digital data is unstructured [1], taking the form of documents, emails, social media posts, images, and videos. In 2020 there are 3.8 billion people using social media, 43% of whom research products on social networks. 52% of online brand discovery takes place through social channels [2]. Knowing what your customers are saying about your brand, and what they are saying about your competition, is critical to building a successful business, and making proper use of the increasing amount of unstructured data is key. Every time your brand is mentioned in a review or social media post offers an opportunity to obtain an insight into your customer's feelings. Machine learning can be used to automate the process of reviewing all this data and extracting key insights that can lead to actionable steps for your business. This analysis can extend to how your brand is viewed compared to your competitors, allowing you to identify weaknesses in your businesses and possible market gaps that you can take advantage of. The automatic tagging of negative messages also enables you to prioritize issues and ensure that the most urgent communications are responded to first. Analyzing key words and phrases will allow you to track where you are doing well or areas in which you need improvement. This sort of analysis will give you a real time look into how your company is being viewed and allow you to measure the impact your company's actions and product releases have on those views.

Rationale Statement

The model that will be the end goal of this project will take any input text and classify it as being a positive, neutral, or negative sentiment. It will then extract the key words or phrases of the text. This type of analysis is currently being used in various fields; ecommerce, marketing, advertising, and politics, just to name a few. The main benefits include the ability to analyze and sort large amounts of text data in real time, in a consistent manner, without the need for a human reader.

Data Requirements and Sources

Building such a model will require a large amount of text data that has been pre-labelled with appropriate classes, such as rating scores. To satisfy this I have elected to work with the Amazon Review Data (2018) resource. This a collection of 233.1 million Amazon reviews, maintained by Jianmo Ni, and updated in 2018 [3]. Use of the dataset is permitted with a citation to the paper, *Justifying Recommendations Using Distantly-Labeled Reviews and Fine-Grained Aspects*, authored by Jianmo Ni, Jiacheng Li, and Julian McAuley [4]. The 233.1 million reviews have been reduced into 5-core and rating-only subsets, and then further divided according to product type, as shown in Figure 1.



"Small" subsets for experimentation

If you're using this data for a class project (or similar) please consider using one of these smaller datasets below before requesting the larger files.

K-cores (i.e., dense subsets): These data have been reduced to extract the k-core, such that each of the remaining users and items have k reviews each.

Ratings only: These datasets include no metadata or reviews, but only (item,user,rating,timestamp) tuples. Thus they are suitable for use with mymedialite (or similar) packages.

You can directly download the following smaller per-category datasets.

Amazon Fashion	5-core (3,176 reviews)	ratings only (883,636 ratings)
All Beauty	5-core (5,269 reviews)	ratings only (371,345 ratings)
Appliances	5-core (2,277 reviews)	ratings only (602,777 ratings)
Arts, Crafts and Sewing	5-core (494,485 reviews)	ratings only (2,875,917 ratings)
Automotive	5-core (1,711,519 reviews)	ratings only (7,990,166 ratings)
Books	5-core (27,164,983 reviews)	ratings only (51,311,621 ratings)
CDs and Vinyl	5-core (1,443,755 reviews)	ratings only (4,543,369 ratings)
Cell Phones and Accessories	5-core (1,128,437 reviews)	ratings only (10,063,255 ratings)
Clothing, Shoes and Jewelry	5-core (11,285,464 reviews)	ratings only (32,292,099 ratings)
Digital Music	5-core (169,781 reviews)	ratings only (1,584,082 ratings)
Electronics	5-core (6,739,590 reviews)	ratings only (20,994,353 ratings)
Gift Cards	5-core (2,972 reviews)	ratings only (147,194 ratings)
Grocery and Gourmet Food	5-core (1,143,860 reviews)	ratings only (5,074,160 ratings)
Home and Kitchen	5-core (6,898,955 reviews)	ratings only (21,928,568 ratings)
Industrial and Scientific	5-core (77,071 reviews)	ratings only (1,758,333 ratings)
Kindle Store	5-core (2,222,983 reviews)	ratings only (5,722,988 ratings)
Luxury Beauty	5-core (34,278 reviews)	ratings only (574,628 ratings)
Magazine Subscriptions	5-core (2,375 reviews)	ratings only (89,689 ratings)
Movies and TV	5-core (3,410,019 reviews)	ratings only (8,765,568 ratings)
Musical Instruments	5-core (231,392 reviews)	ratings only (1,512,530 ratings)
Office Products	5-core (800,357 reviews)	ratings only (5,581,313 ratings)
Patio, Lawn and Garden	5-core (798,415 reviews)	ratings only (5,236,058 ratings)
Pet Supplies	5-core (2,098,325 reviews)	ratings only (6,542,483 ratings)
Prime Pantry	5-core (137,788 reviews)	ratings only (471,614 ratings)
Software	5-core (12,805 reviews)	ratings only (459,436 ratings)
Sports and Outdoors	5-core (2,839,940 reviews)	ratings only (12,980,837 ratings)
Tools and Home Improvement	5-core (2,070,831 reviews)	ratings only (9,015,203 ratings)
Toys and Games	5-core (1,828,971 reviews)	ratings only (8,201,231 ratings)
Video Games	5-core (497,577 reviews)	ratings only (2,565,349 ratings)

Figure 1: Amazon Review Data (2018) Subsets

For the purposes of this project I have decided to work with the "Arts, Crafts and Sewing" 5-core dataset. This is a relatively large subset of the overall dataset, offering 494,485 reviews. Along with a rating and the text and summary of the review, the dataset also provides a reviewer name and ID, the number of times the review has been voted as being helpful, and other identifying features. A sample of the data is shown in Figure 2.

```
df = pd.read_json('Arts_Crafts_and_Sewing_5.json.gz', orient='records', lines=True)
df.shape
```

```
(494485, 12)
```

```
df.head()
```

	overall	verified	reviewTime	reviewerID	asin	style	reviewerName	reviewText	summary	unixReviewTime	vote	image
0	4	True	03 29, 2016	AIE8N9U317ZBM	0449819906	{'Format:': ' Kindle Edition'}	Zelmira, Ph.D.	Contains some interesting stitches.	Four Stars	1459209600	NaN	NaN
1	5	True	08 12, 2015	A3ECOW0TWLH9V6	0449819906	{'Format:': ' Paperback'}	Dangerous when Cooking	I'm a fairly experienced knitter of the one-co...	My current favorite go-to guide for inspiration	1439337600	18	NaN
2	4	True	04 5, 2015	A278N8QX9TY2OS	0449819906	{'Format:': ' Paperback'}	Just us	Great book but the index is terrible. Had to w...	lots of great examples, good instructions, col...	1428192000	3	NaN
3	5	True	10 11, 2014	A123W8HIK76XCN	0449819906	{'Format:': ' Kindle Edition'}	Amazon Customer	I purchased the Kindle edition which is incred...	Another little gem by Melissa Leapman	1412985600	NaN	NaN
4	5	True	05 8, 2014	A2A6MZ2QB4AE0L	0449819906	{'Format:': ' Paperback'}	Sustainability	Very well laid out and very easy to read.\n\nT...	Very comprehensive	1399507200	NaN	NaN

Figure 2: Sample of the "Arts, Crafts and Sewing" 5-core Dataset

Data Limitations, Assumptions, Constraints

This project will aim to analyze real Amazon reviews written by real human beings, and I make the following assumptions before looking into the text data:

- The text may or may not contain punctuation
- The text will probably contain spelling errors
- The text may contain special characters, urls, emojis
- The tone of the text will vary – there may be issues with sarcasm and comparisons, for example

I also assume that the distribution of product ratings will be extremely imbalanced. It would be a personal hypothesis that a customer is more likely to review a product that they absolutely loved or hated, rather than if it were merely adequate.

The model I create will be limited by its ability to process aspects of human speech and text that would be easy for a human interpreter to understand. The model will be constrained by its ability to recognize tone and sarcasm, for example. The model must also be able to examine word combinations when assigning labels. For example, looking just at the keyword "bad" might indicate a negative sentiment, but if the review actually said "not bad" then the score would be different. A way to deal with this is by using n-grams, which will split the text into word combinations rather than single words. This will allow a phrase like "not bad" to be considered as a single entity. Context and comparisons will also be difficulties. A phrase such as "this is better than other products" is certainly positive, but "this is better than nothing" is negative.

Exploratory Data Analysis

This project will focus on predicting the ratings or sentiment based on the review text, and so the first thing I do is remove all unnecessary columns and the reviews which are blank. Next, I examine the ratings. A plot of the distribution reveals that 5/5 comprises most of the ratings, as shown in Figure 3.

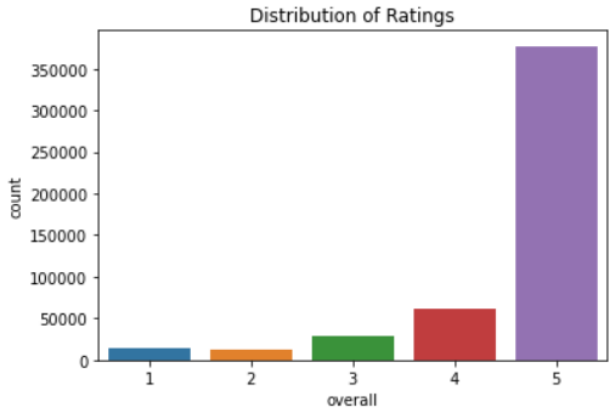


Figure 3: A Plot of the Distribution of Class Labels

Generating the counts and percentages for each class label further illustrates the imbalance this dataset contains, as shown in Figure 4.

```
# Get the rating counts
df['overall'].value_counts()
```

```
5    377313
4     61127
3     28622
1     14353
2     12781
Name: overall, dtype: int64
```

```
5    76.348858
4    12.368979
3     5.791629
2     2.586221
1     2.904313
Name: overall, dtype: float64
```

Figure 4: Count and Percentages of Class Labels

This imbalance will have to be addressed in the algorithm testing and evaluation stage of the project. The reason for this, as seen in Figure 4, is that over 88% of all reviews are given a positive rating of 4 or 5. This means that an algorithm that simply predicts positive for every review will achieve an accuracy score of 88%. There are several approaches that can be taken to address this. Two common techniques are over-sampling and under-sampling. In over-sampling instances of the smaller classes are duplicated in the training dataset. This can overcome imbalance but can lead to overfitting. Under-sampling is removing instances of the larger classes, but this creates the opportunity to lose potentially useful information contained in the removed rows. A third solution would be data augmentation [5]. Since the Amazon Review dataset consists of numerous product categories, I can work with one, split it into training and testing data, and then augment the training set with reviews from a secondary dataset, thereby balancing the distribution of the classes.

To address this imbalance while also keeping the dataset to a workable size I have employed the technique of under-sampling the majority classes. This process will be explored in a subsequent section of this report.

I now examine the actual review text. First, I create a column that counts the number of words contained within each row.

```
# For each row, split the review into words and get the count
df['num words'] = df['reviewText'].apply(lambda x: len(str(x).split()))
```

```
# Get the summary statistics
df['num words'].describe()
```

```
count    494196.000000
mean         30.978948
std          55.670248
min           0.000000
25%           5.000000
50%          15.000000
75%          35.000000
max        3739.000000
Name: num words, dtype: float64
```

Figure 5: Word Count Statistics

I note that some reviews contain zero words, and these I found to be empty strings. These will be removed later as part of the data cleaning process. To get a sense of the reviews I first examine the shortest reviews – those containing only a single word.

```
df[df['num words'] == 1]
```

	overall	reviewText	num words
34	5	Nice	1
63	4	gift	1
64	5	ok	1
89	4	Good	1
112	5	gift	1
...	...	...	...
494403	5	PERFECT!	1
494422	5	Good	1
494429	5	Beautiful	1
494436	5	good	1
494473	5	Excellent	1

27511 rows × 3 columns

```
df[(df['num words'] == 1) & (df['overall'] == 1)]
```

	overall	reviewText	num words
5134	1	Yarn	1
10513	1	NOTHIN	1
11136	1	Ripoff	1
13227	1	Whatever	1
14367	1	ok	1
...	...	...	...
474397	1	Worthless.	1
480377	1	Junk	1
484340	1	Cheap	1
485594	1	cheesy	1
487698	1	PERFECT	1

169 rows × 3 columns

Figure 6: Samples of One Word Reviews

Figure 6 give a sense of these one-word reviews for different labels. I note that the word is mostly very predictive of the rating, although one person has written “PERFECT” and given a one out of five rating. Next, I examine the distribution of the word length for the different rating classes. To do this I examine just reviews shorter than 100 words. The plot can be observed in Figure 7. Shorter reviews are more common for all the rating classes.

```
sns.displot(df_short, x="num words", hue="overall", kind='kde')
<seaborn.axisgrid.FacetGrid at 0x132edccbbe0>
```

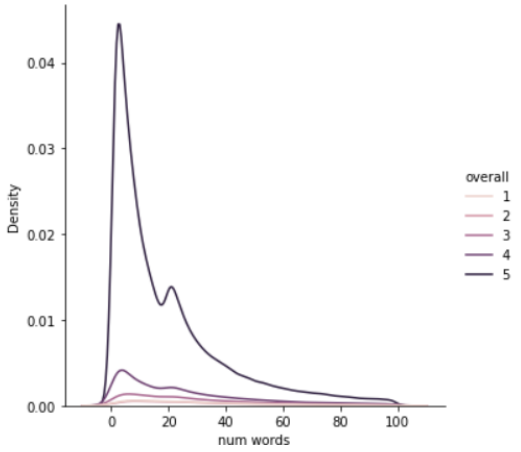


Figure 7: Word Count by Rating (Reviews < 100 words)

Generating some statistics for this I find that the average length of a review tends to increase as the rating decreases. This is something I would have assumed to be true. People who are greatly dissatisfied with a product will be more likely to take the time to write a longer review, either to vent their frustrations or to alert others to their negative experiences.

```
Five Star Review Lengths:
Shortest: 0
Longest: 3445
Average: 26.96077262113948

Three Star Review Lengths:
Shortest: 1
Longest: 2070
Average: 45.28271958633219

One Star Review Lengths:
Shortest: 1
Longest: 1250
Average: 48.056364523096214
```

Figure 8: Word Count Statistics

The final stage of my Exploratory Data Analysis dealt with examining the most common words, both in the dataset as a whole and in the three classes I will be looking to predict – positive, neutral, and negative. To do this I made use of Scikit Learn's Count Vectorizer. This splits the text into individual words, or word combinations, and then generates a matrix where each column represents a unique word, and each row represents one review. The values in the matrix are the number of times that word appears in that review. Using this I was able to count the occurrences of each unique word or word combination and then sort these values.

Performing this on the entire review set revealed all of the most common words are "stopwords"; common English words that are not predictive, such as "the", "and", "to", "it" and "for". These will be removed as part of the data cleaning process. Ignoring these for now revealed the most common words as being "great", "use", "love", "good", "just", and "like". Because the reviews are over 88% positive it was not a surprise that these sorts of words would be very common.

I now examine the most common words and two-word combinations (bigrams) in reviews of each of the three classes. I display these results here as Word Clouds, where the size of the word is representative of its prevalence in the subset of reviews that have positive, neutral, or negative sentiment. I see that generally the bigrams are much more indicative of the sentiment than the single words are.

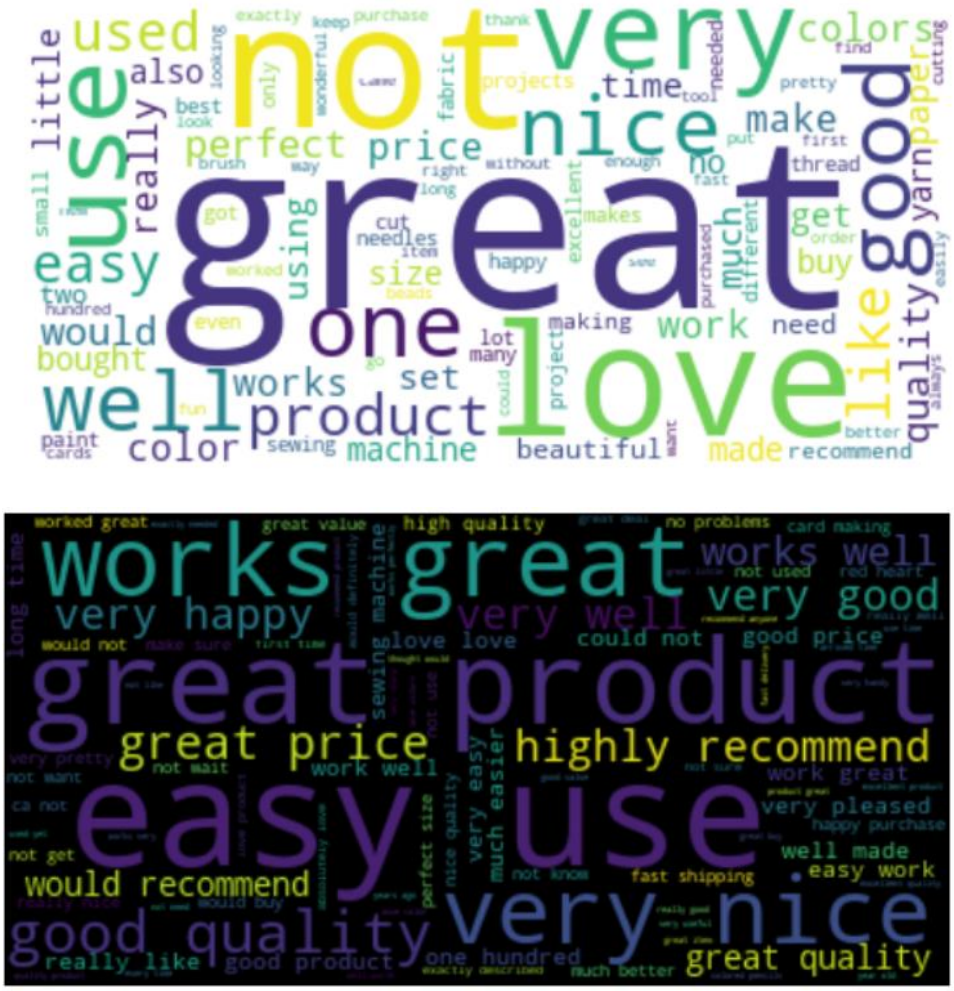
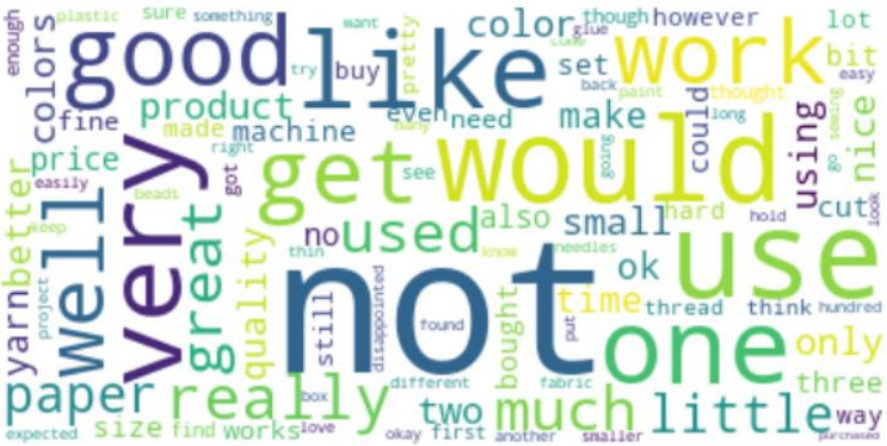


Figure 9: Five Star Reviews – Top 100 Unigrams and Top 100 Bigrams



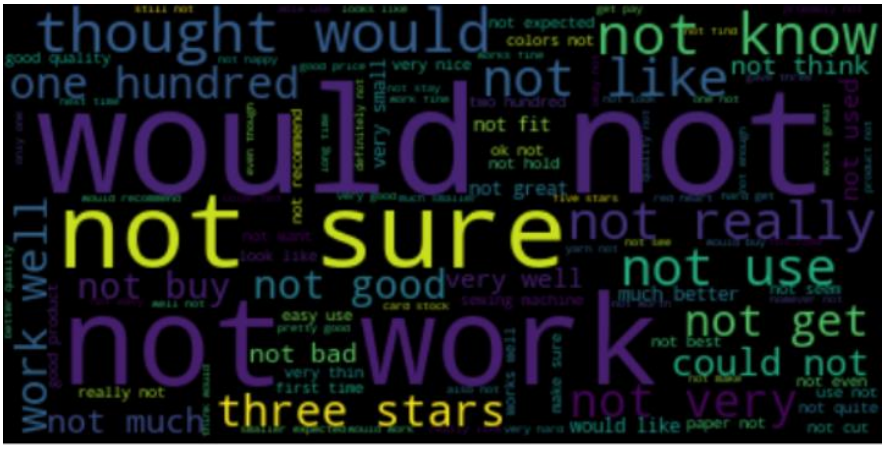


Figure 10: Three Star Reviews – Top 100 Unigrams and Top 100 Bigrams

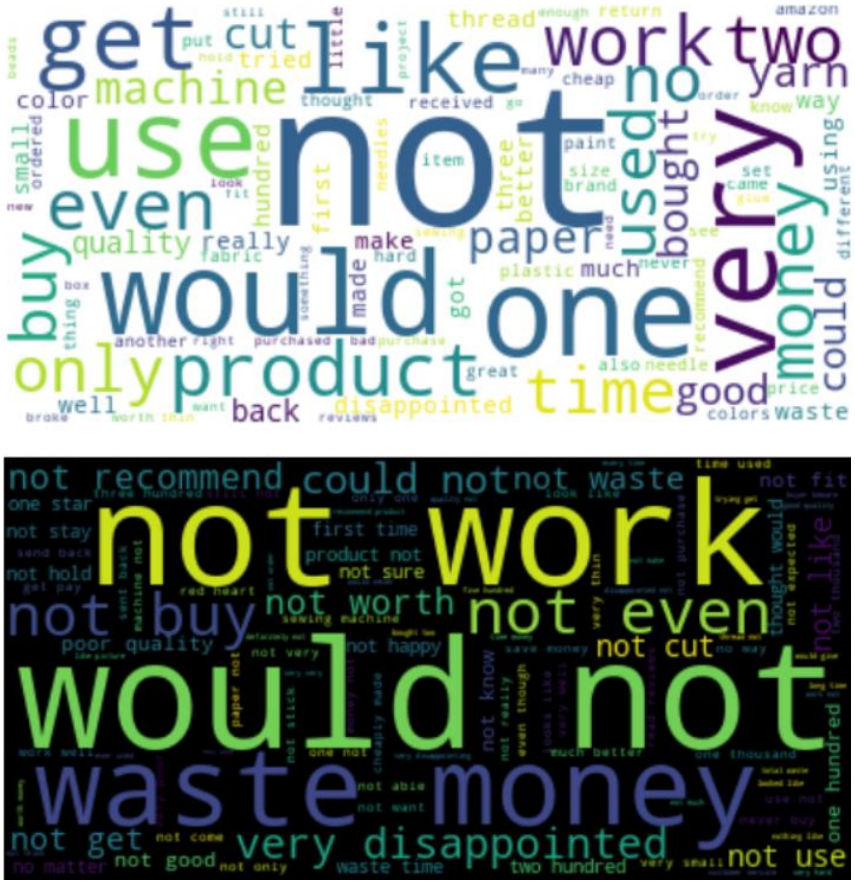


Figure 11: One Star Reviews – Top 100 Unigrams and Top 100 Bigrams

Feature Engineering and Text Preparation

As previously described, since I am dealing with unstructured text data there is much that must be done to simplify and clean the data prior to any algorithm testing. This is my process for addressing the data assumptions, limitations, and constraints, and I perform this in eight stages:

1. Expand contractions
   Using Python's Regular Expressions I replace "won't" with "will not" and "can't" with "can not". For other contractions I substitute "not", "are", "is", and so on for the 't, 're, 's.

```
def decontract(sentence):
    sentence = re.sub(r"won\'t", "will not", sentence)
    sentence = re.sub(r"can\'t", "can not", sentence)

    sentence = re.sub(r"n\'t", " not", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"\'s", " is", sentence)
    sentence = re.sub(r"\'d", " would", sentence)
    sentence = re.sub(r"\'ll", " will", sentence)
    sentence = re.sub(r"\'t", " not", sentence)
    sentence = re.sub(r"\'ve", " have", sentence)
    sentence = re.sub(r"\'m", " am", sentence)
    return sentence
```

Figure 12: Expanding Contractions

2. Remove line breaks
   I use Regular Expressions to locate all "\n" occurrences and replace them with a single space.

```
def remove_linebreaks(input):
    text = re.compile(r'\n')
    return text.sub(r' ',input)
```

Figure 13: Remove Line Breaks

3. Remove punctuation
   I use Python's string.punctuation to create a list of punctuation characters and then remove these from the text.

```
# Get a list of punctuation characters
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
def remove_punctuation(input):
    # Check each character against the punctuation list
    no_punc = [char for char in input if char not in string.punctuation]
    # Rejoin the non-punctuations characters back into their original words
    no_punc = ''.join(no_punc)
    return no_punc
```

Figure 14: Remove Punctuation and Special Characters

4. Convert numbers to text
   Num2words [6] is a package that automatically converts digits into text. 9 becomes "nine" and 42 becomes "forty-two", for example.

```python
def replace_numbers(text):
    words = []
    # split the text into words
    for word in text.split():
        # if the word is a number, convert it to text
        if word.isdigit():
            words.append(num2words(word))
        else:
            words.append(word)
    # convert the text back into a string with words separated by spaces
    return " ".join(words)
```

Figure 15: Convert Numbers to Text

5. Remove punctuation again
   Since num2words adds dashes, I run my function again to remove these.
6. Lowercase all text
   I convert all text to lowercase using .lower().
7. Remove stopwords
   Finally, I use the Natural Language Toolkit [7] to import a list of English stopwords. I examine this list and determine to remove a few – "no", "not", "only" and "very". I feel that these words can be related to sentiment and therefore should not be removed. I remove all other stop words from my review set to complete the text manipulation for now (I will make use of Stemming during the modelling phase of the project).

```python
def remove_stopwords(input):
    # Split the input, checking each word against the stop words list
    no_stop = [word for word in input.split() if word not in stopwords_list]
    # Return the cleaned text
    no_stop = ' '.join(no_stop)
    return no_stop
```

Figure 16: Remove Stopwords

8. Remove empty reviews
   Now I search for all reviews that have become just an empty string as a result of the cleaning process. A review that was simply "A+", for example, would have had the plus sign removed as punctuation and the "a" removed as a stop word. Removing these rows and resetting the index results in a fully cleaned dataset.

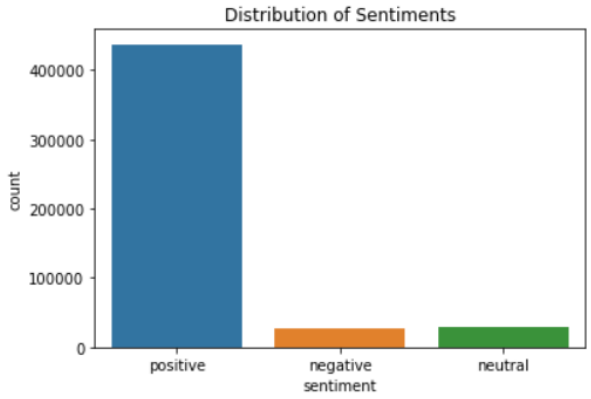
	overall	reviewText
0	4	contains interesting stitches
1	5	fairly experienced knitter onecolor color bloc...
2	4	great book index terrible write high light cro...
3	5	purchased kindle edition incredibly handy part...
4	5	very well laid very easy read book also nice s...
...	...	...
494191	4	not love price watercolor pens not messy score
494192	5	lots color markers
494193	5	really fun use love watercolor not know made w...
494194	1	box says vibrant colors only pencils actually ...
494195	3	core very hard not transfer color paper well p...

493265 rows × 2 columns

Figure 17: Cleaned Dataset

Creating the Sentiment Column

Before testing algorithms I must create the target classes. I convert ratings of four and five to "positive", three to "neutral", and one and two to "negative". I again note the imbalance in the classes.




```
positive    88.701003
neutral      5.798911
negative     5.500086
Name: sentiment, dtype: float64
```

Figure 18: Percentages of Class Labels in Cleaned Dataset

Splitting the Data and Resampling the Training Set

To prepare for the evaluation of learning algorithms I first split the data into training and testing sets. I will be under-sampling the majority class in the training data by a large percentage, and so I split the data first 95/5 in favour of training. This will result in a split that is close to the normal 80/20 once the under-sampling is complete. I begin with a training dataset consisting of 468,601 reviews, with a distribution of:

```
Positive reviews: 415680
Neutral reviews: 27164
Negative reviews: 25757
```

Figure 14: Class Distribution in Training Data

I make use of Scikit Learns' resample to balance the classes to match the size of the negative reviews set. I am left with 77,271 rows for training and 24,664 rows for testing. The training classes are now balanced.

```
positive    25757
negative    25757
neutral     25757
Name: label, dtype: int64
```

Figure 19: Class Distribution in Training Data after Under-Sampling

Selecting an Evaluation Metric

For this multiclass classification problem I now have an even distribution of classes within my training data, and so I elect to use accuracy score as a performance measure due to its ease of interpretability. I will also use the more robust Precision and Recall and F1 Score to measure the performance of the model on the various classes.

Testing Process

Beyond the evaluation metrics described above there are two methods by which I will be able to evaluate the model's performance. First, the algorithm I end up focussing primarily on is Logistic Regression. Using this model, I can generate the coefficients for each of the three class labels. I can match these against the Count Vectorizer's matrix to extract the words or phrases that correspond to these weights. Using this process, I generate the terms that the model is weighing most heavily in predicting the class labels. I examine these to see if anything stands out.

Next, having these lists in hand I will test the model on text that does not contain these most influential terms. This will give me a good understanding of how well the model is performing. To do this I will make use of the predict_proba() function to generate the probabilities for each of the labels. I will be able to determine how certain the model is of its prediction when faced with text where the sentiment is not so obvious.

Feature Extraction

As previously explained, I make use of the Count Vectorizer to transform the raw text into a form that is suitable for machine learning algorithms. There are alternatives such as TF-IDF, Word2Vec, and GloVe, which aim to capture more meaning of the words, but using the Count Vectorizer allows me to test models under different arrangements – using only single words, using word combinations, or using a combination of the two.

Modelling Techniques

I have selected six classification algorithms to test: Decision Tree, Random Forest, Logistic Regression, Support Vector Machines, Naïve Bayes, and k-Nearest Neighbours. And I have selected four approaches: using only single words (unigrams), using only bigrams, using unigrams and bigrams together, and finally using unigrams, bigrams and trigrams together. After the first two approaches I apply stemming to the text to see if it improves the accuracy.

1. Unigrams Only:
   I fit the Count Vectorizer on the training data and create a sparse matrix 77272 by 39376, indicating there are 39,376 unique words in the training set. I transform the training and testing data with the vectorizer and measure the accuracy of the six algorithms on both the training and testing data.

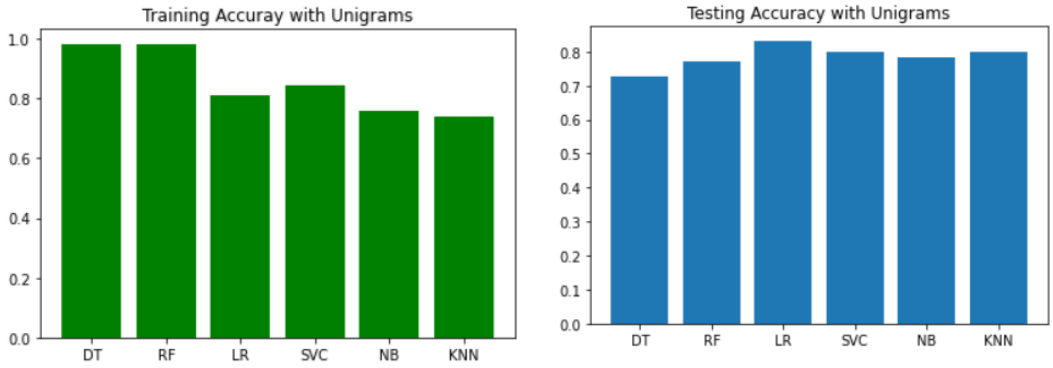


Figure 20: Accuracy using Unigrams

All models perform similarly on the testing data, though Logistic Regression was the most accurate. Decision Trees, Random Forest, and Support Vector Machines all took significantly longer to train than the rest of the models.

2. Bigrams Only:
   I re-vectorize the training data and get a vocabulary of 681,378 unique bigrams. I fit and train the same six algorithms and get the following testing accuracies.

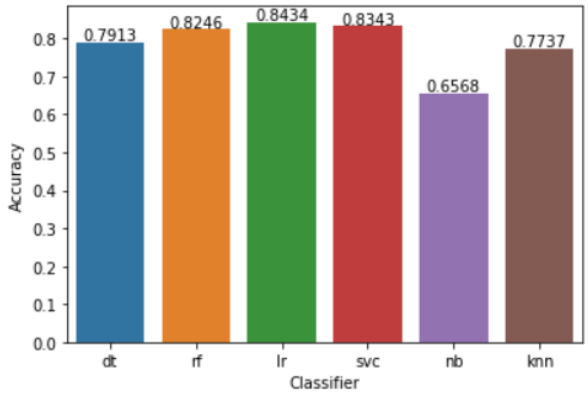


Figure 21: Accuracy using Bigrams

Once again Logistic Regression performs the best. Naïve Bayes did not perform well but the others were not far behind Logistic Regression. From this point I decided to focus my efforts on Logistic Regression for two reasons. First, it has shown the best performance to this point and, second, it trains much faster than some of the other algorithms, even on large datasets. I now apply stemming to measure its impact on the results.

3. Bigrams with Stemming:
Stemming simplifies the data by reducing plural words to their singular form and generally shortening words to their roots. To perform this I make use of NLTK's Snowball Stemmer [8], which I apply to both the training and testing data.

```
def stem_text(input):
    # Create a stemmer object
    stemmer = SnowballStemmer('english')

    #Split the text and use the stemmer on each word
    text = input.split()
    words = ""
    for i in text:
        words += (stemmer.stem(i))+' '

    # Return the recombined text
    return words
```

Figure 22: Stemming

An example of stemming is shown in Figure 18.

```
# Print a review to test
clean_df['reviewText'][0]
```

```
'contains interesting stitches'
```

```
# See the result after stemming
stem_text(clean_df['reviewText'][0])
```

```
'contain interest stitch '
```

Figure 23: An Example of Stemming

I vectorize the stemmed data and refit the Logistic Regression model, achieving a training accuracy of 0.9498 and a testing accuracy of 0.8509.

4. Unigrams and Bigrams:
Using the stemmed data, I fit a Count Vectorizer to consider both unigrams and bigrams together. I end up with a vocabulary length of 584,152, which is a significant reduction in

size. The Logistic Regression Classifier now achieves a training accuracy of 0.9635 and a testing accuracy of 0.8577. Both are slight improvements.

5. Unigrams, Bigrams, and Trigrams:
   Lastly, I vectorize the training data according to unigrams, bigrams, and trigrams, and create a vocabulary with a size of 1,753,074 tokens. The Logistic Regression Classifier achieves a training accuracy of 0.9741 and a testing accuracy of 0.8607. Once again this is a very small improvement in both.

6. Hyperparameter Tuning:
   I test several combinations of solvers, multi-class parameters, and C values to see if I can improve on this result. All of the Logistic Regression accuracies are summarized in the following table.

	Vectorizer	Solver	Multi Class	C	Testing Accuracy
Logistic Regression	Unigrams	Liblinear	Auto	1	0.8328
Logistic Regression	Bigrams	Liblinear	Auto	1	0.8434
Stemming Added					
Logistic Regression	Bigrams	Liblinear	Auto	1	0.8509
Logistic Regression	Unigrams + Bigrams	Liblinear	Auto	1	0.8577
Logistic Regression	Unigrams + Bigrams + Trigrams	Liblinear	Auto	1	0.8607
Logistic Regression	Unigrams + Bigrams + Trigrams	Liblinear	One Vs Rest	1	0.8607
Logistic Regression	Unigrams + Bigrams + Trigrams	Newton-CG	One Vs Rest	1	0.8509
Logistic Regression	Unigrams + Bigrams + Trigrams	Newton-CG	Multinomial	1	0.8517
Logistic Regression	Unigrams + Bigrams + Trigrams	Liblinear	One Vs Rest	0.5	0.8542
Logistic Regression	Unigrams + Bigrams + Trigrams	Liblinear	One Vs Rest	0.1	0.8590

Figure 24: Logistic Regression Results

Lowering the C value increases regularization and reduces overfitting on the training data. In going from C = 0.5 to C = 0.1 the testing accuracy increases, but the training accuracy decreases from 0.932 to 0.8560. For this reason I did not further lower the C value.

The best model found was Logistic Regression with the One Vs Rest parameter and the liblinear solver, using the combination of unigrams, bigrams, and trigrams. In this multiclass problem One Vs Rest functions by fitting a binary classifier to each of the class labels. I will next examine this chosen model.

Model Examination

I first check the classification report, which is reproduced here.

	precision	recall	f1-score	support
negative	0.54	0.79	0.64	1373
neutral	0.30	0.66	0.41	1440
positive	0.99	0.88	0.93	21851
accuracy			0.86	24664
macro avg	0.61	0.78	0.66	24664
weighted avg	0.92	0.86	0.88	24664

Figure 25: Classification Report of Logistic Regression Model

The model is doing very well on the positive class, reasonably well on the negative class, and not very well on the neutral class. The high difference between precision and recall for the neutral class means the model is predicting a lot more labels as neutral than it should be.

The Final Model

```
# As noted, I will use a Count Vectorizer with a combination of unigrams, bigrams, and trigrams
# This combination produced the highest accuracy in the algorithm testing notebook

vectorizer = CountVectorizer(ngram_range=(1,3))

# Vectorization
X_train_cv = vectorizer.fit_transform(X_train)
X_test_cv = vectorizer.transform(X_test)
```

```
# Fitting the Logistic Regression classifier and checking the accuracies
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train_cv, y_train)
lr_train_preds = lr.predict(X_train_cv)
lr_preds = lr.predict(X_test_cv)
lr_train_acc = accuracy_score(y_train, lr_train_preds)
lr_test_acc = accuracy_score(y_test, lr_preds)
print("Training Accuracy:", lr_train_acc)
print("Testing Accuracy:", lr_test_acc)
```

```
Training Accuracy: 0.974078244101927
Testing Accuracy: 0.8607281868310087
```

Figure 26: The Final Model

Examining Keywords and Phrases

As previously described, I can use the vectorizer's features and the model's coefficients to produce lists of the terms with largest coefficients for each of the classes.

```
# The number of features
len(vectorizer.get_feature_names())
```

```
1753074
```

```
# The number of coefficients
lr.coef_.shape
```

```
(3, 1753074)
```

```
# The classes
lr.classes_
```

```
array(['negative', 'neutral', 'positive'], dtype=object)
```

Figure 26: Vectorizer Features

```
feature_to_coef = {
    word: coef for word, coef in zip(
        vectorizer.get_feature_names(), lr.coef_[0])
    }
```

```
# Sort the created dictionary according to value and print the five largest
for most_important_negative in sorted(feature_to_coef.items(), key=lambda x: x[1], reverse=True)[:5]:
    print (most_important_negative)
```

Figure 27: Matching Vectorizer Features and Model Coefficients

```
('junk', 3.0726358223736043)
('useless', 2.7383975515160652)
('no good', 2.5882426497003044)
('horribl', 2.5234650915169516)
('terribl', 2.3688897595392047)
```

Figure 28: Largest Coefficients for the Negative Class

```
('three star', 4.0288150141789885)
('okay', 2.350508982553373)
('ok', 2.1440595823248274)
('averag', 2.0967954420772332)
('not bad', 2.0171706545013692)
```

Figure 29: Largest Coefficients for the Neutral Class

```
('awesom', 3.0363366564732814)
('perfect', 3.0031275090763097)
('love', 2.8913258130581916)
('excel', 2.8235221812581006)
('never disappoint', 2.524678366389776)
```

Figure 30: Largest Coefficients for the Positive Class

Prototyping the Model

I created a pipeline to produce results for a user's text. I first combine all of the text cleaning functions, which I named, process_input.

```
def process_input(input):
    input = decontract(input)
    input = remove_linebreaks(input)
    input = remove_punctuation(input)
    input = input.lower()
    input = replace_numbers(input)
    input = remove_punctuation(input)
    input = remove_stopwords(input)
    input = stem_text(input)
    return input
```

Figure 31: Cleaning User Text Function

Next, I define two functions. The first function takes input, processes and vectorizes it using the Count Vectorizer fit on the training data (with ngrams equal to one, two, and three), and then predict the class label and the probabilities for each of the three labels.

```
predict_sentiment()
Enter Your Text Here:

I was looking to get back into painting and these paints were incredible! I was unsure of whether to purchase or not because they were mu
ch more affordable than some other options I saw but I am so glad I purchased these. Whether these are for children or for yourself, you
can enjoy and you really get the feel that they are high quality. I've worked with expensive acrylic paints before and these are very sim
ilar. They blend very seamlessly together, and the colour selection is absolutely gorgeous. The paint tubes are actually a decent size so
you have good value there as well. None of the paints were dried up or anything, and so far they have exceeded my expectations. They arri
ved in very nice packaging as well so I would imagine that it would make a great gift. Highly recommend this!!


The Predicted Sentiment is:  POSITIVE

Analysis:

Probability of Negative Label: 0.0 %
Probability of Neutral Label: 0.13 %
Probabilty of Positive Label: 99.87 %
```

Figure 32: Sample Prediction

The second function does the same as the first but then also extracts the key words or phrases from the text. To accomplish this I create three dictionaries. These contain all of the tokens of the vectorizer, along with their corresponding coefficients in the logistic regression model for the given predicted label,

```
# The coefficients for the negative class
negative_coef = {
    word: coef for word, coef in zip(
        vectorizer.get_feature_names(), lr.coef_[0])
    }

# The coefficients for the neutral class
neutral_coef = {
    word: coef for word, coef in zip(
        vectorizer.get_feature_names(), lr.coef_[1])
    }

# The coefficients for the positive class
positive_coef = {
    word: coef for word, coef in zip(
        vectorizer.get_feature_names(), lr.coef_[2])
    }
```

Figure 33: Matching Text Tokens to Model Coefficients by Class

Next, I make a list of all the tokens in the input text and use the created dictionary for the predicted sentiment. I extract the coefficient for each of the tokens and then sort them.

```
def predict_sentiment_with_analysis():
    # Prompt the user for text
    user_text = input('Enter Your Text Here: \n\n')
    # Process
    processed_text = [process_input(user_text)]
    # Transform the text according to the Count Vectorizer that has been fit
    vec_text = vectorizer.transform(processed_text)

    # Predict the class label
    predicted_sentiment = lr.predict(vec_text)
    # Generate the probabilities for the labels
    confidence = lr.predict_proba(vec_text)

    # Get the features of the input text and store the English words or phrases
    phrases = []
    for item in vec_text.indices:
        phrases.append(vectorizer.get_feature_names()[item])

    # Create a dictionary of the model's coefficients for these features
    importances = dict()
    for phrase in phrases:
        if predicted_sentiment == 'negative':
            importances[phrase] = negative_coef.get(phrase)
        elif predicted_sentiment == 'neutral':
            importances[phrase] = neutral_coef.get(phrase)
        elif predicted_sentiment == 'positive':
            importances[phrase] = positive_coef.get(phrase)
```

Figure 34: Pipeline to Generate Predictions from User Text

I then print a summary of the results which will be shown in the next section of this report.

Testing the Model on Real Reviews

I test the model on a positive, a neutral, and a negative review found on Amazon.

1. Positive Review: 5/5 rating
   (Source: https://www.amazon.ca/product-reviews/B07XP1CNRW/ref=acr_dp_hist_5?ie=UTF8&filterByStar=five_star&reviewerType=all_reviews#reviews-filter-bar)

```
predict_sentiment_with_analysis()

Enter Your Text Here:

I was looking to get back into painting and these paints were incredible! I was unsure of whether to purchase or not because th
ey were much more affordable than some other options I saw but I am so glad I purchased these. Whether these are for children o
r for yourself, you can enjoy and you really get the feel that they are high quality. I've worked with expensive acrylic paints
before and these are very similar. They blend very seamlessly together, and the colour selection is absolutely gorgeous. The pa
int tubes are actually a decent size so you have good value there as well. None of the paints were dried up or anything, and so
far they have exceeded my expectations. They arrived in very nice packaging as well so I would imagine that it would make a gre
at gift. Highly recommend this!!


The Predicted Sentiment is:  POSITIVE

Analysis:

Probability of Negative Label: 0.0 %
Probability of Neutral Label: 0.13 %
Probabilty of Positive Label: 99.87 %

The Five Most Important Stemmed Words or Phrases Are:

       Word or Phrase  Model Coefficient
First           great           2.518729
Second           glad           1.994195
Third        gorgeous           1.860844
Fourth          enjoy           1.394096
Fifth            nice           1.380592
```

Figure 34: Test on a Positive Review

The model predicted positive with 99.87% certainty. This is because this review contained some very positive words – “great”, “glad” and “gorgeous”.

2. Neutral Review: 3/5 rating
   (Source: https://www.amazon.ca/product-reviews/B076KCGRS6/ref=?ie=UTF8&filterByStar=three_star&reviewerType=all_reviews&pageNumber=1#reviews-filter-bar)

```
predict_sentiment_with_analysis()
```

```
Enter Your Text Here:

These pencils are pretty good. They're not amazing but they're good enough for beginners at a decent price. The leads are quite
strong. I've been using them for a few weeks now and I haven't had one break yet. What bothers me is that they aren't labeled,
so it can be hard to tell which color is which. I often have to hold it up into the light to tell the difference, and I still h
ave a hard time since some of them look so similar. I'll usually just scribble it on a separate piece of paper beforehand, just
to be sure I have the correct pencil. In the future, I plan on labeling them myself to avoid wasting so much time. The variety
of colors are okay, but they're not the most "natural" of hues (think comic book colors) I had to go out and purchase additiona
l colors separately (because I needed more pastel colors) Artists might need a different gray pencil because it looks more like
a sandy beige...and the white pencil does absolutely nothing. Overall I am happy with this purchase...but as they run out indiv
idually, I will probably replace each of them one by one with different brands, just so I can choose the exact colors I need fo
r specific projects.


The Predicted Sentiment is:  NEUTRAL

Analysis:

Probability of Negative Label: 0.0 %
Probability of Neutral Label: 99.99 %
Probabilty of Positive Label: 0.01 %

The Five Most Important Stemmed Words or Phrases Are:

       Word or Phrase  Model Coefficient
First            okay           2.350509
Second         decent           1.185844
Third       hard time           0.874930
Fourth         probabl           0.758386
Fifth     good enough           0.607285
```

Figure 35: Test on a Neutral Review

Once again the model performs extremely well. Here we see a number of words and phrases that clearly represent a neutral feeling – “okay”, “decent” and “good enough”. To try something more difficult I have found a negative review that does not contain any extremely negative words or phrases. There is no “terrible”, “horrible”, “waste of money”, for example.

3. Negative Review: 1/5 rating
   (Source: https://www.amazon.ca/product-reviews/B076KCGRS6/ref=cm_cr_unknown?ie=UTF8&filterByStar=one_star&reviewerType=all_reviews&pageNumber=1#reviews-filter-bar)

```
predict_sentiment_with_analysis()
```

```
Enter Your Text Here:

We ordered the full 3 sets and before we knew it they were breaking constantly. Thinking it was our sharpener, we used a differ
ent one and the problem persisted. You will get breakages but not losing 50mm in less than a hour. Before long that pencil will
be gone. We returned ours for a full refund and bought some Castles, much better. Don't know how these are constructed, but it
looks like a 2 piece outer with pressured stain, which disguises the grain whether good or bad. We contacted the seller who was
very sympathetic and sent us a free box of Cobras, but sadly no different, actually these were worse.


The Predicted Sentiment is:  NEGATIVE

Analysis:

Probability of Negative Label: 82.56 %
Probability of Neutral Label: 17.44 %
Probabilty of Positive Label: 0.0 %

The Five Most Important Stemmed Words or Phrases Are:

       Word or Phrase  Model Coefficient
First          return           1.895161
Second            bad           1.473265
Third            sent           1.338053
Fourth           wors           1.110683
Fifth             sad           1.070498
```

Figure 36: Test on a Negative Review

The certainty isn't as high as in the two previous cases but the model still correctly predicts the class label with an almost 83% probability. I note here the words it has returned as keywords – "return", "bad", "sent", "worse" and "sad". Overall, the model is performing very well.

Deployment

Now that I had my model built and tested I was ready to deploy it as a full functioning application. To accomplish this I make use of Flask. I first pickle both my fit Count Vectorizer and Logistic Regression Model. I then import these into a Flask app and recreate my pipeline for user data. I use the same process here as I did in my Jupyter Notebook. I take the input text, process and vectorize it, and then use the model to generate its prediction and probabilities.

In terms of output I want two things – the predicted sentiment and a summary of the model's prediction. I create and display a bar graph of the probabilities for the three sentiment classes. Next, I use HTML and CSS to style and create my app. I have created a simple graphic and styled my application. The elements will resize and maintain their proportions on any size window.

To deploy the application I have used Heroku. It is active and available at:

https://sentiment-analyzer-mm.herokuapp.com/

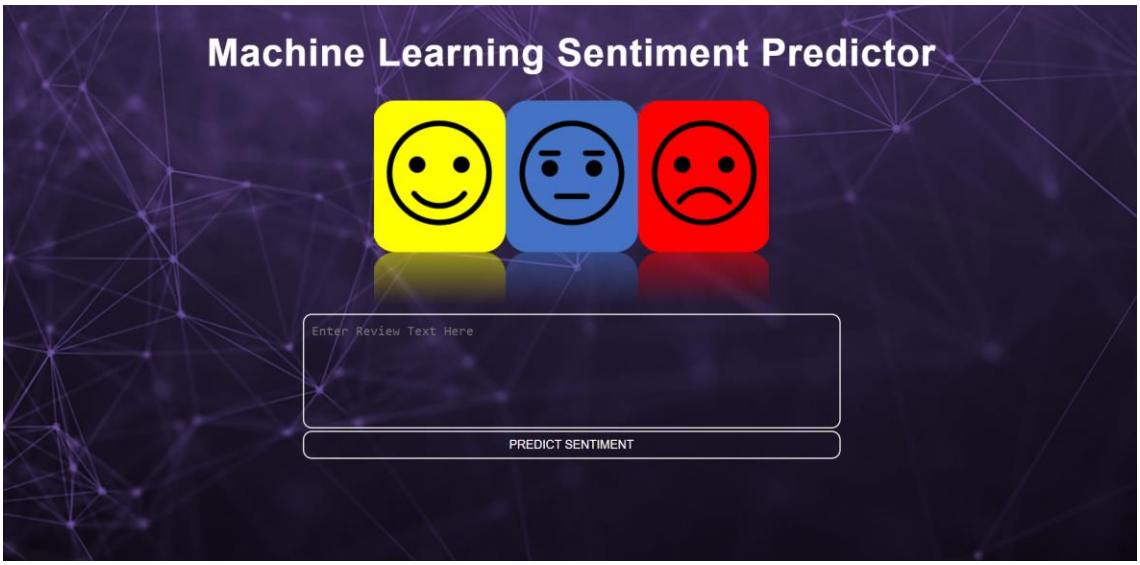


Figure 37: Deployment Landing Page

Upon pasting text and pressing the button, the output is displayed as shown here.

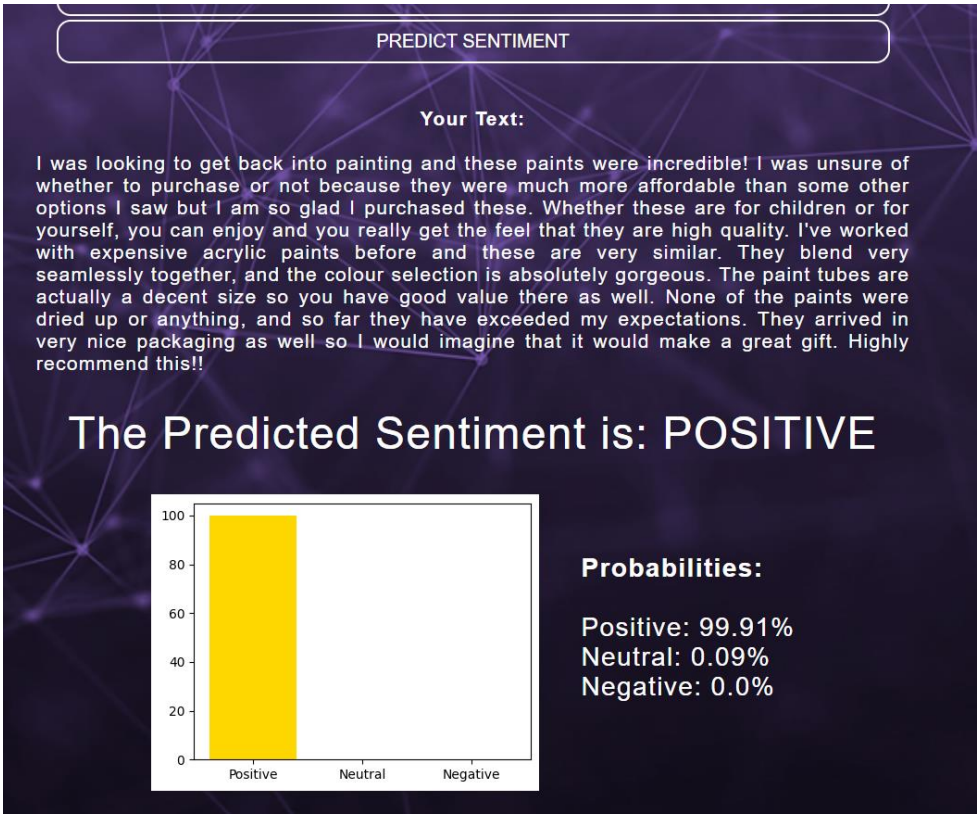


Figure 38: Sample Output

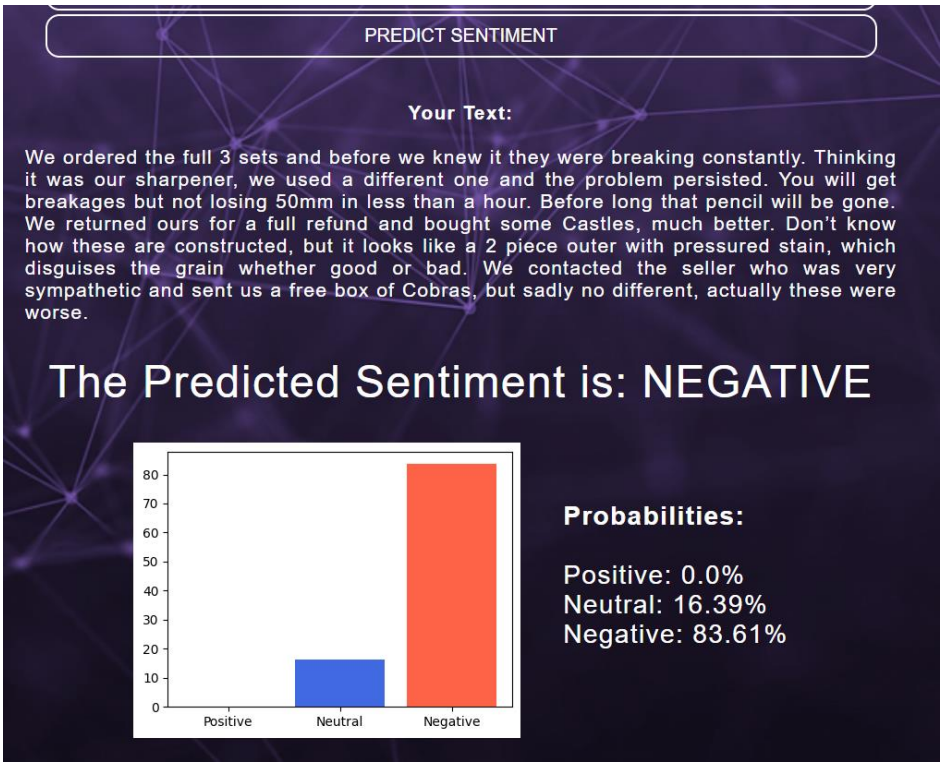


Figure 39: Sample Output

Project Timeline

The project is now complete. To summarize my deliverables:

Folder 1 – Business Understanding and Problem Discovery

Statement of Work

Folder 2 – Data Acquisition and Understanding

Notebook 1 (EDA)

Notebook 2 (Data Preparation)

Version 2 Report

Folder 3 – ML Modeling and Evaluation

Notebook 3 (Model Preparation and Testing)

Notebook 4 (Sentiment Analysis and Key Phrases on User Text)

Folder 4 – Delivery and Acceptance

Flask App, HTML Index, and associated files

Deployment Document

Final Report

Presentation Slides

Conclusion

I have used Machine Learning to deliver a fully functional and accurate Sentiment Analysis tool that can be accessed online. This application will generate sentiment predictions for any pasted text. You can also paste multiple reviews for a product or company and predict an overall sentiment for it.

Task	Planned Completion By
Phase 1: Data Analysis	
Exploratory Data Analysis	November 8
Data Cleaning and Manipulation	November 15
Statistical Analysis	November 15
Feature Extraction	November 15
Update SOW	November 15
Phase 2: Modelling	
Test and Evaluate Learning Algorithms and Frameworks	November 15
Research Software Tools and Pipeline	November 22
Assess Data Assumptions, Limitations, and Constraints	November 22
Phase 3: Prototyping	
Prototype Proposed Model Architecture	November 22
Model Tweaking and Evaluation	November 22
Update SOW	November 22
Phase 4: Delivery	
Develop Software Pipeline to Host Model	December 18
Develop Solution Endpoint for Final User	December 18
Prepare Final Report and Documentation	December 18

Figure 40: Project Milestones and Timelines

References

[1] Davis D. AI Unleashes the Power of Unstructured Data. CIO. https://www.cio.com/article/3406806/ai-unleashes-the-power-of-unstructured-data.html. Accessed October 28, 2020.

[2] Cooper P. 140+ Social Media Statistics that Matter to Marketers in 2020. Hootsuite. https://blog.hootsuite.com/social-media-statistics-for-social-media-managers/. Accessed October 30, 2020.

[3] Ni J. Amazon Review Data (2018). https://nijianmo.github.io/amazon/index.html. Accessed October 28, 2020.

[4] Ni J, Li J, McAuley J. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. *Empirical Methods in Natural Language Processing (EMNLP)*. 2019. http://cseweb.ucsd.edu/~jmcauley/pdfs/emnlp19a.pdf. Accessed October 28, 2020.

[5] Brownlee J. 8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset. Machine Learning Mastery. https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/. Accessed October 30, 2020.

[6] num2words 0.5.10. PyPi. https://pypi.org/project/num2words/. Accessed November 26, 2020.

[7] Natural Language Toolkit. NLTK 3.5 Documentation. https://www.nltk.org/. Accessed October 30, 2020.

[8] Stemmers. NLTK. https://www.nltk.org/howto/stem.html. Accessed November 26, 2020.