Objective of project is to perform sentiment analysis (positive / negative) on Amazon Product Reviews.

1. Preprocessing

Preprocessing plays an important role in text mining. The purpose of this step is to convert raw data into value data which is ready for modelling. In this project, we used the package "tm" to clean texts.

- Remove punctuation: All punctuation marks according to the priorities should be dealt with.
- Remove number and white spaces: white spaces and number give no information, we removed them to save the memory.
- Transform low case: Convert every word into low cases to handle low cases and high cases in the same way.
- Remove stop words: These words are frequent but do not provide information. Beside 174 stop words on the common list, we removed other words such as "ipod" and "player". Since the raw data are reviews on products, the frequency of these words is high, but they do not carry too much meaning.
- Stemming: We removed stem of words to reduce the number of words and have accurate matching stems.
- Remove Sparse terms: To reduce the dimension of the Document-Term Matrix, we removed the less frequent terms.

In the preprocessing step, the challenge is that we had to examine raw data and identify words that are frequent but give little meaning in predicting sentiments. These words will be added to the stop words and removed. This process reiterates until the words we obtained eventually are meaningful.

2. N-gram analysis

N- Gram is conducted to use co-occurring words for analysis. For Bi-Gram, it takes every pair of words, moving one word forward. N-Gram can take as many co-occurring words as needed, forming expressions that can make more sense than taking words only individually.

For example, "service" or "customer" alone may not make a difference to classify good from bad reviews, but "customer service" might. Taking into account N-Gram in our model therefore seems more accurate than unigram.

We implemented 2 different models using N-Gram:

- Bi-Gram: we considered only pair of words.
- Unigram/Bigram combination: we considered words alone plus their bigrams.

To use N-gram, we had to change several elements of the model compared to unigram:

- *Dictionary*: We completed the dictionary of positive and negative words with the 'not',' no', 'nor' and 'neither' words in combination with negative single words. Then we added those bigrams to the positive dictionary vector. We did the same for negative dictionary. It resulted in almost 12K unigrams and bigrams in each dictionary, instead of the 2K we have initially.

- *Preprocessing*: We customized the stop words we had initially, by removing the negative words like "didn't", so negative combination would remain for bigram analysis.
- *Create NGram DTM :* we used the Rweka package, to create first a bigram only, then a unigram plus bigram combination.
- *Modeling:* We had to tune the Sparsity in the modelling process, has to be optimized differently as there are three times more features using unigram plus bigram.

After running the 2 models, the most accurate approach in minimizing False Positive rate, was the Unigram plus Bigram combination as we expected, because it takes more relevant words in our classification problem. For example, 2 bigrams that appeared to have a high predictability in positive reviews are "battery Life" and "easy use", which are an addition to "usb" and "easy" unigrams.

3. Term Frequency – Inverse Document Frequency (TFIDF)

We have applied Term Frequency – Inverse Document Frequency on our terms to determine the importance of each term in helping to predict the positive or negative sentiment. Multiplying TF and IDF values generated from below formulas will give us the TFIDF value of each term in each document.

	Positive terms	Negative terms
TF	$\sum_{i=1}^{3000} \frac{\text{Frequency count of terms in positive document } i}{\text{Total number of terms in positive document } i}$	$\sum_{i=1}^{3000} \frac{\text{Frequency count of terms in negative document } i}{\text{Total number of terms in negative document } i}$
IDF	$1 + \ln\left(\frac{\text{Total number of negative documents}}{\text{Number of negative documents containing the positive term}}\right)$	$1 + \ln\left(\frac{\text{Total number of positive documents}}{\text{Number of positive documents containing the negative term}}\right)$

- **Term Frequency (Normalized) – TF**

Counting frequency of the terms in each document enables us to differentiate between how many times a word is used and the count is normalized by the number of terms in that document. The more often the positive terms occurs in its positive document, the greater the importance of the term in a positive document and vice versa. The same word will have different TF value in different documents.

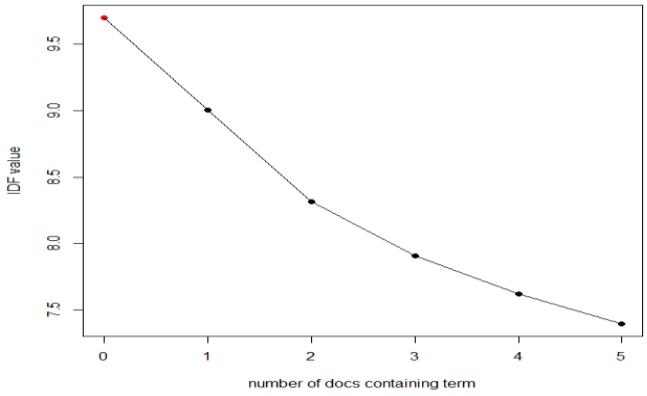
- **Inverse Document Frequency - IDF**

IDF measures sparseness of a term (eg. Positive) in other (eg. Negative) documents. A positive term frequently occurring in negative documents is likely not useful for classification. Therefore, the more often the positive term occurs in negative documents, the lesser the importance of that term in a positive document and vice versa. The same word will only have one IDF value.

- **Challenges**

The calculation of IDF poses a problem when the number of negative documents containing a positive term is 0 and vice versa. This will cause the IDF value to be infinity and introduces NAs into the dataset. To overcome this problem, we decided to do a linear extrapolation of the IDF value from points 1 and 2 to obtain a dummy IDF value (red point) when the number of documents containing the term is 0.

We applied this dummy IDF value of 9.70 as IDF for positive terms that did not appear in negative documents at all and vice versa. This allows us to put the most importance on these terms.



4. Model Building and Evaluation

With the base table built from dictionary, n-gram and TFIDF, we have explored 5 machine learning models – Logistic Regression, Support Vector Machine, Naïve Bayes, Neural Network and Decision Tree. Out of the 5 models, Support Vector Machine has provided us with the highest AUC of 0.976 and a low false positive rate of 0.0288. Neural Network has similar performance with a slightly lower AUC of 0.973 but a lower false positive rate of 0.0255.

	Logistic Regression	Support Vector Machine	Naive Bayes	Neural Network	Decision Tree
Accuracy	0.962	0.976	0.361	0.973	0.824
Specificity	0.966	0.971	0.492	0.975	0.756
Precision	0.965	0.971	0.310	0.974	0.785
Sensitivity	0.958	0.980	0.229	0.972	0.892
False Positive Rate	0.0344	0.0288	0.508	0.0255	0.244

5. Desiderata

Our group created a list regarding the needs to better increase the value of text mining for Amazon: Some of the uses of text mining should involve live streaming for product reviews to initiate recommendations based on the sentiment analysis. It would be beneficial to also explore neutral sentiments of each review to have a better outlook on each product. Additionally, it would be valuable to detect fake reviews to ensure the sentiment analysis does not skew to any direction. Although a relatively large dataset is being used, it would also be advantageous to use a larger data for higher accuracy as Amazon has over millions of reviews. Also, it is important to determine the correlation between the written reviews, and reviews like star rating. One of the other issues that is faced is the calibration of the reviews. For example, the star rating can have different weightage in different countries and knowing this would increase the accuracy for each sentiment. Additionally, using different machine learning techniques and models to sort out sentiments would be beneficial to help the accuracy of the models. Lastly, the models should be tested on other companies reviews for comparison between their products.