



BT4221: Big Data Techniques and Technology
Food Recommendation System

Members:

Rachael Tay Sze Min (A0156726X)

Tanya Ramesh (A0162732J)

Liew Mei Xin (A0171703M)

Oon Kei Zee Casey (A0171354J)

Yue Jing Ying (A0171154M)

1. Introduction	3
1.1 Problem Statement	3
1.2 Motivation	3
2. Dataset	3
2.1 Data Rationale	3
2.2 Data Collection	4
3. Data Preprocessing	5
4. Exploratory Data Analysis	7
4.1 Distribution of Product Rating Scores	7
4.2 Word Cloud for Product Reviews	7
5. Modelling	8
5.1 Phase 1 - Simple Recommender using CNN	8
5.1.1 Feature Preparation	8
5.1.2 1-Dimensional Convolutional Neural Network (CNN)	9
5.2 Phase 2 - Item-Based Recommendation System	12
5.2.1 Overview - Recommendation Process	12
5.2.2 Cosine Similarity	13
5.2.3 Shortlisting the products	14
5.2.4 Determine if shortlisted product is recommended using Neural Network	15
5.3 Comparison	16
6. Recommendations	17
7. Use Cases	18
7.2 Suicide Prevention	18
7.3 Spam Mail	19
7.5 Medical Diagnosis	19
8. Conclusion	19
9. References	20

1. Introduction

1.1 Problem Statement

More people are increasingly engaging in ecommerce and online shopping. Ecommerce platforms want to build customer loyalty and encourage more buying behaviour. A recommendation system seeks to predict how a user would rate an item and recommend products that are likely to receive favourable ratings. Through recommendation systems, companies retain customers by capitalising on their preferences and extending their interests to other products, making customers more likely to stay and continue using their platforms. After all, many ecommerce platforms such as Amazon take a commission of the sales. The more they sell, the more money they earn.

Due to the intangible nature of ecommerce, both customers and platforms want to assure the quality of products, thus platforms often provide an avenue for customers to review products, whether they liked it, or what they found flawed. From the text of reviews, we aim to understand how much people liked a product and what they liked about it. From there, we match reviews and qualities of other products and then recommend these products to those who are interested.

1.2 Motivation

In this project, we have selected food reviews from Amazon to use. The methods and experiences we gained may be applied to other forms of sentiment analysis as the processes behind them are similar. Identifying suicide notes and hate speech can be explored with the model that we made.

2. Dataset

2.1 Data Rationale

With a majority of the data available being unstructured, there is a large potential to do data mining to glean insights to be able to drive better data-driven decision making for businesses.

The Amazon Fine Foods dataset was selected to understand people's preferences via reviews and to create a recommendation system that allows them to explore new options. Food is a

relatively wide topic with many different types of food catering to all types of taste. It is a great starting point to build a model which can be further diversified into.

2.2 Data Collection

The data was sourced from a dataset on Kaggle which consisted of consolidated customer reviews about food products that were bought on Amazon. The data was collected by the Stanford Network Analysis Project and it was consolidated into one sqlite database consisting of reviews up to October 2012. The data was then downloaded as a csv file to make it appropriate for preprocessing and modelling. There are approximately 500,000 instances in the dataset where each record contains details about an individual review.

Field Name	Description	Data Type
Id	Row Id	Numeric
ProductId	Unique identifier for the product	String
User Id	Unique identifier for the user	String
ProfileName	Profile Name for the user	String
HelpfulnessNumerator	Number of users who found the review helpful	Integer
HelpfulnessDenominator	Number of users who indicated whether they found the review helpful or not	Integer
Score	Rating between 1 and 5	Integer
Time	Timestamp for the review	Integer
Summary	Brief summary of the review	String
Text	Brief summary of the review	String

Table 1: Data Fields

```
food_df = pd.read_csv('./drive/My Drive/BT4221_Big Data/food_df.csv')
food_df = food_df.drop('Unnamed: 0', axis=1)
food_df.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dli pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Figure 1: Dataframe Illustrating Data in the Dataset

3. Data Preprocessing

After importing the csv file as a pandas dataframe, we checked and found 16 and 27 reviews with missing *ProfileName* and *Summary* values respectively. We filled these missing values with empty strings. Next, we removed reviews that are duplicates. Lastly, we only kept reviews with *HelpfulnessNumerator* values that are less than or equal to their *HelpfulnessDenominator* values. This is because the number of users who found a review helpful cannot be more than the number of users who indicated whether they found the review helpful or not.

To prepare the reviews for conversion into word embeddings, we performed text preprocessing to only keep important words for better result accuracy and faster computation:

Symbols

Remove symbols like ,&#\!?;></ as they are not meaningful and will only interfere with analyses.

Lowercasing

Convert all uppercase letters to lowercase for consistency and ease of processing. For example, the sentence "Product Received As Advertised" will become "product received as advertised".

Tokenization

Break sentences up into its individual words. For example, the sentence "product received as advertised" will be tokenized to ['product', 'received', 'as', 'advertised']. This is so that the importance of each word can be determined based on its frequency.

Stop Words Removal

Stop words are commonly occurring words such as "I", "a", "is" and "the", which are meaningless and should be filtered out. This is done by comparing tokens to a list of stop words from the NLTK library. If a token is not found in the list of stop words, it is not a stop word and thus,

should be taken. Otherwise, it is a stop word and should be left out. For example, after removal of stop words, ['product', 'received', 'as', 'advertised'] will become ['product', 'received', 'advertised']. By removing low information words, we can focus on more important words instead.

Lemmatization

Lemmatization is the reducing of inflected words to their root forms. For example, 'helps', 'helping', 'helped' will all be lemmatized to 'help'. This is necessary to ensure that words with the same root form are not interpreted as different words. We wrote a function to handle lemmatization with the use of spaCy's 'en_core_web_sm' model, where we only kept tokens with 'NOUN', 'ADJ' and 'VERB' part-of-speech tags. This is because such words give better information and thus, are more meaningful.

We chose lemmatization over stemming as it produces more accurate results. Stemming simply removes prefixes and suffixes from words such that the reduced word may not be an actual word in the dictionary (e.g. "troubling" to "trouble"). Also, stemming is unable to relate words of different forms but with the same basic meaning (e.g. "better" to "bet"). On the other hand, the reduction of words through lemmatization will always be an actual word (e.g. "troubling" to "trouble") and lemmatization is capable of identifying words of different forms but with the same basic meaning (e.g. "better" to "good"). Hence, even though lemmatization is slower, we still chose it over stemming as we value accuracy over speed.

Short Words Removal

Remove words that are less than 3 characters in length as it is highly unlikely that such short words are significant.

4. Exploratory Data Analysis

4.1 Distribution of Product Rating Scores

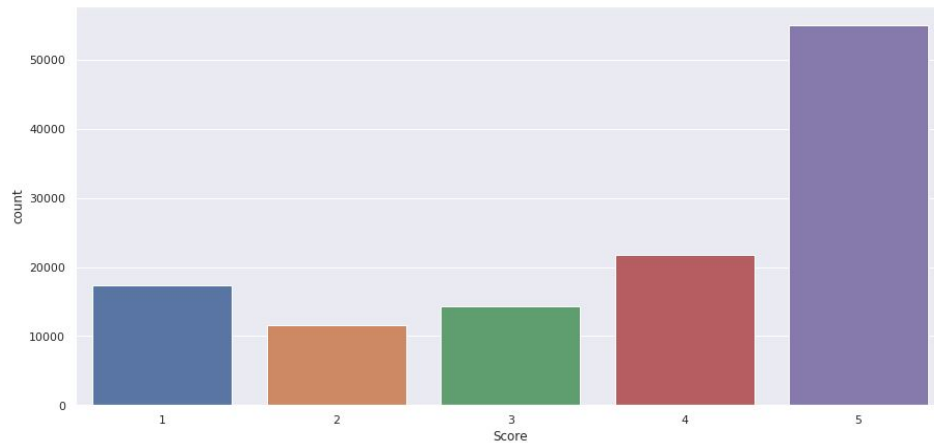


Figure 2: Graph of Product Rating Scores

To gauge how well-perceived Amazon fine food products are on average, we plotted a graph to visualise the distribution of *Score* values across all products. From Figure 2 above, we can observe that most products received a rating of 5, indicating that a majority of customers were satisfied with the products they purchased and received.

4.2 Word Cloud for Product Reviews



Figure 3: Word Cloud for Product Reviews

We generated a word cloud, as seen in Figure 3 above, based on words that appeared in reviews left by customers. This gives us an overall picture of words commonly included in

reviews, which tells us what customers usually talk about in their reviews. From the word cloud, we can see that some of the more frequently used words include 'look', 'taste' and 'good'.

5. Modelling

5.1 Phase 1 - Simple Recommender using CNN

5.1.1 Feature Preparation

For our Phase 1 neural network model, its aim is to predict if a product should be recommended or not based on product reviews and helpfulness ratings: Reviews are provided by users which consist of positive and negative ones for each product. For helpfulness ratings, a product is likely to be recommended if the reviews pertaining to the product are helpful. Thus for each product, a derived feature of *HelpfulnessPercentage* was created, whose values are calculated from the division of *HelpfulnessNumerator* values by *HelpfulnessDenominator* values. This derived feature is useful as it is more reflective of helpfulness as compared to absolute values.

Since the same product could receive multiple reviews and thus each have its own helpfulness rating, before we do any computations, we aggregate each individual review into one for each product. For helpfulness rating, the helpfulness numerator and denominator is then summed up accordingly before it is divided to obtain average helpfulness rating across all reviews. The average review scores are also computed in a similar fashion.

With average review scores, we can then create a new column for the output target - the *Recommend?* column which represents the prediction classes of our model, where 1 indicates that the product will be recommended (positive review) while 0 indicates that it will not be recommended (negative review). This is because we want to recommend products that are perceived positively by other users. Hence, if review score is more than 3, the product will be recommended while those with reviews scores less than 3 will not be recommended.

	ProductId	Text	HelpfulnessPercentage	Recommend?
0	B001E4KFG0	buy several vitality can dog food product find...	1.0	1
1	B00813GRG4	product arrive label jumbo salt peanutsthe pea...	0.0	0
2	B000LQOCH0	confection century light pillowy citrus gelati...	1.0	1
3	B000UA0QIQ	look secret ingredient robitussin believe find...	1.0	0
4	B006K2ZZ7K	great taffy great price wide assortment yummy ...	0.0	1

Figure 4: Dataset after Data Transformation

Before text is passed into a neural network, it must be converted into vectors or sequence as input. Each word in review is assigned an integer using *Tokenizer.text_to_sequences()* and that integer is placed in a list. We set the maximum term frequency as 30,000 - only the top 29,999 words that appear most frequently will be given an integer. For example, if we have a sentence “buy several vitality can dog”, each word is assigned a number. We suppose buy=1, several=2, vitality=3, can=4 and dog=5. After *texts_to_sequences* is called, our sentence will look like [1, 2, 3, 4, 5] with each integer assigned to each unique word.

As all the sentences must have the same input shape, we pad the sentences. Due to time and memory constraints, we set our *max_sequence_length=1000* during padding using *pad_sequences()*. After padding our sentence will be [0, 0, 0,, 1, 2, 3, 4, 5, 6]. 0 means the word does not exist in this particular review. The total number of tokens for each padded sentence will be 1000.

5.1.2 1-Dimensional Convolutional Neural Network (CNN)

The word embedding matrix is then combined with *HelpfulnessPercentage* column before they are fed into a neural network. Here, the type of neural network we used for phase 1 is Convolutional Neural Network (CNN), which is commonly used in text classification tasks and often have good results as compared to simple neural networks. Let us briefly see what happens when we use CNN on text data through a diagram (Figure 5). The result of each convolution will fire when a special pattern is detected. By varying kernel sizes and concatenating their outputs, patterns of multiples sizes (2, 3, or 5 adjacent words) can be detected. Patterns can be expressions like “I hate”, “very good” and therefore, CNNs can identify them in the sentence regardless of their position.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1001, 50)	7958950
dropout (Dropout)	(None, 1001, 50)	0
conv1d (Conv1D)	(None, 999, 250)	37750
global_max_pooling1d (Global	(None, 250)	0
dense (Dense)	(None, 250)	62750
dropout_1 (Dropout)	(None, 250)	0
activation (Activation)	(None, 250)	0
dense_1 (Dense)	(None, 1)	251
activation_1 (Activation)	(None, 1)	0
Total params: 8,059,701		
Trainable params: 8,059,701		
Non-trainable params: 0		

Figure 5: 1D-Convolutional Neural Network Parameters

We split the input data into training and test dataset by ratio of 8:2, which result in 54,099 training samples and 13,525 testing samples. The training matrix is passed to embedding_layer with arguments as follows: total size of vocab = 159,178 (the number of unique words+1) , dimensions = 50, length of input vector = 1,001 (length of vector = 1000 + HelpfulnessPercentage column). The output from the Embedding layer will be 159,178 vectors of 50 dimensions each. The embedding layer learns an embedding for each of the words in the vocabulary and will output a 2-dimensional vector of embeddings for each word in the review. CNN is then trained using a total of 250 filters with size 3 with GlobalMaxPooling1D layers are applied to each layer. A dropout layer and then final dense layer is applied. Since the output target is binary, the loss function is binary cross entropy and activation function used in the output layer is sigmoid function.

```

Build model...
Train on 43279 samples, validate on 10820 samples
Epoch 1/5
43279/43279 [=====] - 1853s 43ms/step - loss: 0.4710 - acc: 0.8078 - val_loss: 0.4021 - val_acc: 0.8350
Epoch 2/5
43279/43279 [=====] - 1790s 41ms/step - loss: 0.3746 - acc: 0.8393 - val_loss: 0.3468 - val_acc: 0.8547
Epoch 3/5
43279/43279 [=====] - 1812s 42ms/step - loss: 0.3410 - acc: 0.8550 - val_loss: 0.3298 - val_acc: 0.8619
Epoch 4/5
43279/43279 [=====] - 1820s 42ms/step - loss: 0.3261 - acc: 0.8604 - val_loss: 0.3298 - val_acc: 0.8624
Epoch 5/5
43279/43279 [=====] - 1808s 42ms/step - loss: 0.3155 - acc: 0.8660 - val_loss: 0.3280 - val_acc: 0.8618

```

Figure 6: 1D CNN Training and Validation Accuracy and Loss

```

[[ 1057, 1553],
 [ 364, 10551]]

```

Figure 7: Confusion Matrix

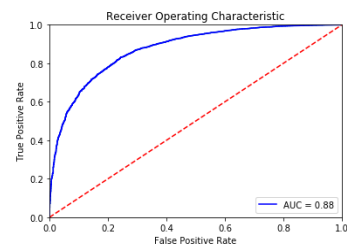


Figure 8: ROC Curve

The training accuracy is 87.1% and testing accuracy is 85.8% which are quite high. The AUC area is 0.88 which implies that CNN performs quite well in classifying recommended and non-recommended samples.

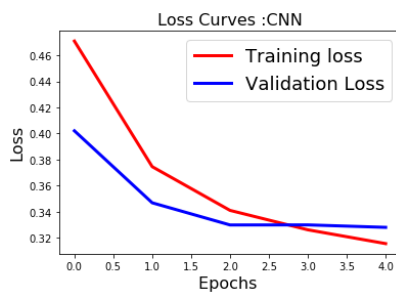


Figure 9: Loss over Epoch

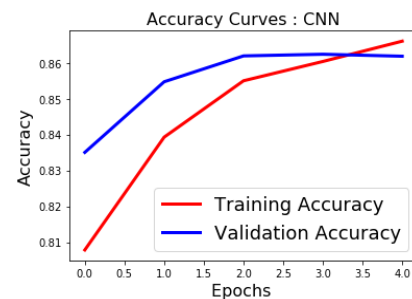


Figure 10: Accuracy over Epoch

As the number of epochs reaches 3, the loss curve and accuracy curve of both training set and validation set intersect which means that it is good to stop training at epoch 3 before they depart consistently.

However, when we examine the confusion matrix, we notice that false positive is very high which means precision is low while recall is high. This means that the model predicts more cases to be recommended while in actual fact they are not recommended. In this context, it is appropriate as in the case when users have already made their purchase decision on the

products and no longer need that particular product, then the algorithm should also recommend other types of products which the user may be interested in.

5.2 Phase 2 - Item-Based Recommendation System

5.2.1 Overview - Recommendation Process

In the dataset, each record is a review for a particular product with an associated score and helpfulness parameters. Since the pipeline is based on the idea of an item-based recommendation system, the recommendations have to be made on a product-level. Hence, we want to build a product recommendation system which acts as a filtering system that seeks to predict and show items that a user would like to consider during their purchases. Given the product ID searched by the user, our model will first filter a list of items similar to the searched product. The list of similar items are then used as input to the neural network model which will then predict which of the products from the input list should be recommended to the user. For example, if a user is searching for product “coffee”, it is found that the products “tea” and “milk” are similar while “spaghetti” and “rice” are not similar to “coffee”. “tea” and “milk” are then fed into our Phase 2 neural network model which will predict if “tea” and “milk” should be recommended to the model based on their respective similarity values and helpfulness percentage.

The process flow diagram in Figure 11 below illustrates the stages of our Phase 2 Neural Network model.



Figure 11: Cosine Similarity and Neural Network Pipeline

In stage 1, the initial dataset which consists of *Text* as reviews, *HelpfulnessPercentage* as independent variables and *Recommend?* as output. The initial dataset is the same input as our Phase 1 Neural Network model as shown in Figure 12 below.

	ProductId	Text	HelpfulnessPercentage	Recommend?
0	B001E4KFG0	buy several vitality can dog food product find...	1.0	1
1	B00813GRG4	product arrive label jumbo salt peanutsthe pea...	0.0	0
2	B000LQOCH0	confection century light pillowy citrus gelati...	1.0	1
3	B000UA0QIQ	look secret ingredient robitussin believe find...	1.0	0
4	B006K2ZZ7K	great taffy great price wide assortment yummy ...	0.0	1

Figure 12: Product Reviews and HelpfulnessPercentage as Input, Recommend? as Output

Next, if given a search product ID, the similarity values of all products are then computed using cosine similarity to measure how similar the products are to the searched product based on their reviews.

5.2.2 Cosine Similarity

Typically, item recommendations are based on the product name. However, since there was no product name included in the dataset, the cosine similarity was run between the reviews of a product that was randomly chosen from the dataset against the reviews of all the other products in the dataset.

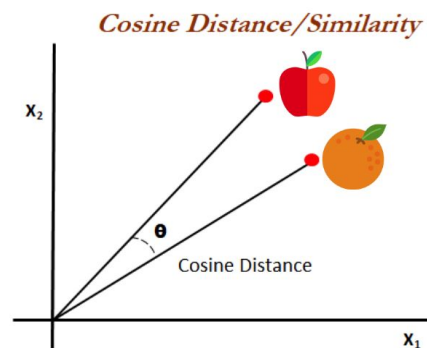


Figure 13: Cosine Similarity between Apple and Orange

Firstly, we vectorise reviews using Gensim model, Word2vec, to convert the words into vectors. For this project, we chose Word2vec over GloVe. Word2vec trains the word on its context (skip-gram) or train the context on the word (continuous bag of words) using a 1-hidden layer neural network. This predictive model requires less memory space as compared to GloVe and it is more intuitive to interpret. Hence, we will be generating entities embedding through word2vec over GloVe.

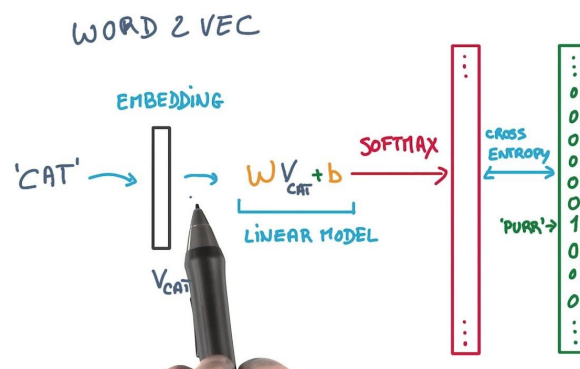


Figure 14: Word2vec Example

Once we form the entities embedding from all the reviews, we calculate the average vector for all words in every sentence in the product reviews. This can be done by tokenizing the input,

which is the product to produce recommendations for, sum the vectors of each word and divide based on the total number of words. This average vector will be the feature vector in which we can conduct cosine similarity on.

Using Scipy's spatial function, we can calculate the cosine distance between these feature vectors, which is between two products' reviews. Lastly, to generate a similarity score, we take one minus this cosine distance.

This process is repeated between the product we would like to produce a recommendation for and other products. For example, if the search product id is "7800648702", the cosine similarity is computed on all products and below figure shows the cosine similarities computed for all products with respect to search product id:

```
[0.7700149416923523,
 0.7817531824111938,
 0.6940114498138428,
 0.769822895526886,
 0.5784621834754944,
 0.6761204600334167,
 0.5222657322883606,
 0.7254374623298645,
 0.7197946906089783,
 0.5038543008501613]
```

Figure 15: List of cosine similarities between the reviews for the product with the productId = '7800648702'

5.2.3 Shortlisting the products

Most item-based recommendation systems recommend a certain number of the most similar products. Thus, they meet a certain threshold in terms of similarity. Hence, after running cosine similarity, a threshold was determined to shortlist products whose inputs would be fed into the Neural Network in the last step of this recommendation pipeline. The threshold was $\mu + 2\sigma$ where μ is the mean of all the similarities and σ is the standard deviation of the similarities. This threshold was chosen as cosine similarity has the property of null distribution and as the number of instances grows large it is well approximated by normal distribution. Here the number of instances are large, hence it is apt to shortlist the products using the $\mu + 2\sigma$ threshold since this is equal to 98% which means that those products with a cosine similarity higher than this threshold are more likely to be recommended based on the search term.

5.2.4 Determine if shortlisted product is recommended using Neural Network

The shortlisted products were then fed into a Neural Network to be predicted as whether they would be recommended or not based on the search product. This is important as even though

some products may possess a high similarity to the search product, they may not necessarily be recommended due to low quality. Hence the inputs to the Neural Network as shown below were the cosine similarity of the shortlisted products and the helpfulness percentage for each of the shortlisted products.

	similarity_values	HelpfulnessPercentage	Recommend?
ProductId			
B0000DJDHK	0.921489	0.560000	0
B000271L90	0.933960	0.888889	1
B0008JEZ9K	0.936091	1.000000	1
B000AY9SR2	0.923890	0.693878	1
B000AYFAO2	0.930879	0.791667	1
...
B006W5WGFE	0.919054	0.386364	1
B007JFMH8M	0.943056	0.891304	1
B007NRQ4O4	0.917611	1.000000	1
B007PA33MA	0.924351	0.939394	1
B007URZ4UW	0.927351	1.000000	1

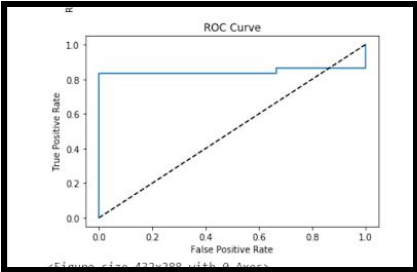
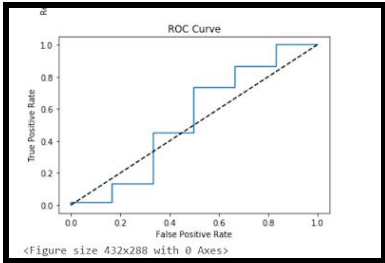
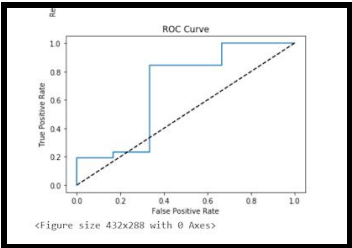
*Figure 16: Descriptive Features for the Neural Network Model.
Recommend? Is the binary target feature*

Our phase 2 neural network model is built using Keras Sequential model with *similarity_values* and *HelpfulnessPercentage* as features and *Recommend?* as the class. Our data is split into 80% training and 20% test. Our model only has 2 hidden layers to reduce overfitting and complexity. Each hidden layer has ReLU activation function to rectify the vanishing gradient problem while the final output layer has Sigmoid activation function, since we are making binary predictions. When compiling our model, we specified the optimizer as Adam and loss function as Binary Cross Entropy, since we are dealing with binary choices. Lastly, we used 'accuracy' as the metrics to judge the performance of our model as we want to determine the percentage of products correctly classified.

5.3 Comparison

Generally, the training and testing accuracy for each searched product are more than 80%. We do notice that the ROC curve is not smooth, which is predictable as this means our model can generally provide discrete predictions rather than a continuous score. This can be remedied by adding more continuous features in the model such as time of reviews to further improve the model. It is also noticed that for some products, there is a high amount of false positive than true positive, which means that the precision is low. This is due to lack of enough data samples fed

into the neural network such that the model have not learnt enough to classify recommended and non-recommended.

'B009SR4OQ2'	<div>Training Accuracy: 94.57%</div> <div>Testing Accuracy: 90.91%</div> <div><div>[[0 3] [0 30]]</div><p>The ROC Curve for 'B009SR4OQ2' shows a True Positive Rate (Y-axis) and False Positive Rate (X-axis) both ranging from 0.0 to 1.0. The curve is a step function that starts at (0,0), jumps to a True Positive Rate of approximately 0.85 at a False Positive Rate of 0, and then jumps to a True Positive Rate of 1.0 at a False Positive Rate of approximately 0.95. A dashed diagonal line represents the performance of a random classifier.</p></div>
'B009SF0TN6'	<div>Training Accuracy: 90.38%</div> <div>Testing Accuracy: 90.91%</div> <div><div>[[0 6] [0 60]]</div><p>The ROC Curve for 'B009SF0TN6' shows a True Positive Rate (Y-axis) and False Positive Rate (X-axis) both ranging from 0.0 to 1.0. The curve is a step function that starts at (0,0), jumps to a True Positive Rate of approximately 0.1 at a False Positive Rate of 0.1, then to 0.4 at 0.3, 0.7 at 0.5, 0.85 at 0.6, and finally to 1.0 at 0.9. A dashed diagonal line represents the performance of a random classifier.</p></div>
'7800648702'	<div>Training Accuracy: 80.80%</div> <div>Testing Accuracy: 87.50%</div> <div><div>[[2 4] [0 26]]</div><p>The ROC Curve for '7800648702' shows a True Positive Rate (Y-axis) and False Positive Rate (X-axis) both ranging from 0.0 to 1.0. The curve is a step function that starts at (0,0), jumps to a True Positive Rate of approximately 0.15 at a False Positive Rate of 0.1, then to 0.25 at 0.2, 0.85 at 0.3, and finally to 1.0 at 0.6. A dashed diagonal line represents the performance of a random classifier.</p></div>

6. Recommendations

Phase 1

Currently the word embedding to 1D-CNN is based on term frequency (most frequent 1000 words will only be taken into account), there could be words which are common but not meaningful to the review is added. TF-IDF word vectors could be the alternative for the word embedding to the model, which measures how important the word to the review. For comparison of models, other types of deep learning models such as RNN, HAN and RCNN can be tested against 1D-CNN to measure which model has the best performance in classifying recommended and non recommended samples. These alternative models are also famously used for text classification.

Phase 2

Considering how different products have different features, it is not a simple task to find recommendations for all. This is especially so if the product is considered an outlier. These outliers would produce a less than ideal recommendation. However, we can consider how recommendation systems are meant to recommend not only based on similarity but based on distinctness so that users will be able to explore more options. This makes recommending something to users complicated. Hence, it is possible that besides considering reviews on an aggregate level, that we can consider the user details as well. An example would be mapping their taste and preference, or even perform clustering to see which other users have similar preferences and recommend things that other users in the cluster have bought but the user themselves did not.

On top of that, since we are using review data, it is important to consider how there are possible fake reviews within the mix. According to ReviewMeta, a website dedicated to detecting fake reviews amongst Amazon reviews, 22.8 million out of 203 million Amazon reviews analyzed are deemed untrustworthy (The Hustle, 2019).

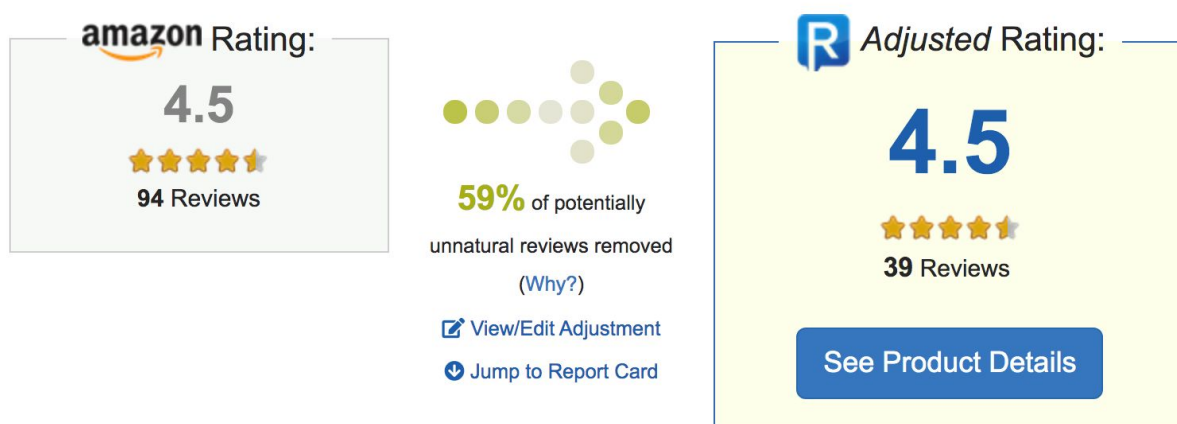


Figure 17: ReviewMeta Adjusted Rating

This potentially influence what kind of recommendations are given, and possibly skew to produce more recommendations for product involved in buying positive fake reviews. Thus, it would improve the accuracy of what should be recommended if we are able to distinguish between real and fake reviews and filter the fake reviews out when building this model.

Dataset Limitations

Item-based recommendation system recommends similar products to a search item that is input by a user, hence similarities have to be accounted for. Typically, item recommendations are based on the product name. However, since there was no product name included in the dataset, this may have affected the accuracy of the product recommendations. Besides, as the products are shortlisted based on the similarity values with respect to the searched product, this results in a small number of data samples for training. To improve, instead of shortlisting the products, we can feed in the entire list of products or choose a large sample from the dataset for training the neural network, to enable the model to learn and generalise well to unknown data.

7. Use Cases

This type of modelling and algorithm can be extended to use cases in other industries, particularly those with text classification purposes and multinomial outcomes:

7.2 Suicide Prevention

Social media platforms like Facebook or Twitter are becoming common channels for people to reveal their suicidal tendencies. Early detection of such ideations can help prevent possible suicide attempts. Suicidal individuals often express helplessness or hopelessness with the use of highly negative language like “kill” or “death”. Through analysing texts of their posts or tweets, converting them into word embeddings and similarly passing them into CNNs as described above, we can predict if these texts are indeed showing suicidal tendencies. If many of such posts or tweets from the same user are suicidal, we can reach out to them and seek help for them before their suicidal thoughts develop further and the unfortunate happens.

7.3 Spam Mail

This model can be integrated into mailboxes for spam detection of incoming emails. The model would require supervised learning with a dataset such that it would have to be able to identify keywords which are commonly associated with spam email. Data sources for this would vary from mailbox to mailbox. In addition, it may require training with other features such as “email” to

identify mail senders that are irrelevant to a particular mail recipient. Wrapping this model in the form of a mailbox-plugin or filter, it can be used to lower mailbox clutter with spam mail.

7.4 Proposal Screening

This model can be used to determine if a particular proposal will get screened or not. Proposal screening can be used in crowdfunding platforms as well as credit scoring when issuing bank loans. The model would require supervised learning as it would have to be able to identify keywords which are commonly associated with approved proposals. Sentiment analysis could also be used to identify the severity and urgency of realising the proposals. The expected training dataset would consist of numeric and text data that could be used to classify if a proposal would be approved or not. Data sources would vary across business domains (i.e data source for bank credit scoring and government school project proposals would differ). This model can be used to reduce bias that may be prevalent during proposal screenings and standardise the screening process.

7.5 Medical Diagnosis

In order for doctors to be able to make better decisions when it comes to diagnosing patients, such predictive methods can play a critical role in improving the accuracy and precision of the diagnosis. One of such ways is training the algorithm to predict, based on images of body scans or symptoms, whether a patient has a certain disease or not. A convolutional neural network in the context of image processing can be used in this use case.

8. Conclusion

Given the evaluation of the models, it can be seen that the Phase 1 model performs better than the Phase 2 model due to large training samples given. From Phase 2, it was evident that more training samples were required and other metadata such as product name and timestamp could have been used. However, the Phase 2 model is better suited to the recommendation system use case than the Phase 1 model.

Text mining could contain useful information for classification problems. However, the large number of word features generated should be reduced with proper features selection techniques to avoid overfitting. The features that would be included in the model would either be determined by the user or by a user-defined threshold.

To optimally design a neural network model, the right activation functions, optimizers and loss functions as well as epochs should be chosen to yield the best results. Since this was a binary classifier, sigmoid was the ideal choice for the activation function. Relu was also used due to its computational efficiency. The loss function used was the binary cross-entropy function which is

well-suited for binary classifiers. Adagrad + Momentum (Adam) optimizer was used due to its computational efficiency.

In tandem to the business use case, more sophisticated refinements could be used to implement these models on an industrial scale. In the specific use case of the Amazon food products, a similar but enhanced model could be used to recommend items not only for food pairs and other product categories as well.

9. References

The Hustle. (2019, Apr 13). 5-star phonies. Inside the fake Amazon review complex. Retrieved from <https://thehustle.co/amazon-fake-reviews>

http://icybcluster.org.ua:34145/technology-documents/cosine_similarity_eng.pdf

Stanford Network Analysis Project. (2017, May 1). Amazon Fine Food Reviews. Retrieved from <https://www.kaggle.com/snap/amazon-fine-food-reviews>.