

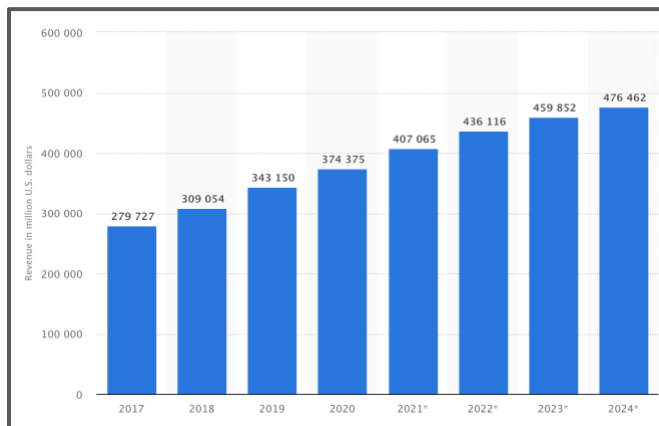
**DATA 255 - Deep Learning**  
**Final Group Project Report**  
**Team 6**  
**December 9th, 2020**

**Sentiment Analysis of Amazon Food Reviews**

Afra Bijli	(Ginny) Jeehee Choi	Hye In Nam	Kimberly Segura
008340631	012565696	014567475	010151609
<a href="mailto:afra.bijli@sjsu.edu">afra.bijli@sjsu.edu</a>	<a href="mailto:jeehee.choi@sjsu.edu">jeehee.choi@sjsu.edu</a>	<a href="mailto:hyein.nam@sjsu.edu">hyein.nam@sjsu.edu</a>	<a href="mailto:kimberly.segura@sjsu.edu">kimberly.segura@sjsu.edu</a>

## 1. INTRODUCTION

E-commerce has become more important and in high demand because of the increase of online shopping post-covid. For example, Amazon is one of the largest online retailers in the world that can efficiently handle high demands of customers during this time. The one of the important factors of a customer's decision to purchase depends on the customer review because it tells an after-use story of products.



**Figure 1. Retail e-commerce sales in the United States from 2017 to 2024** (*in million U.S. dollars*)[1]

The use of sentiment analysis on Amazon review datasets are beneficial in 3 ways.

- (1) Increase online sales
- (2) Improve business prospects, and
- (3) Maintain excellent customer satisfaction

The goal of this project is to study correlation between product values and scores through sentiment analysis.

## 2. DATASET

To perform user review sentiment analysis for online stores, we collected Amazon fine food review data from Kaggle (data source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>). There are 3 criteria to select the data. First, the data should be real-world data. The Amazon fine food reviews are collected from October 1999 to October 2012. Second, the data should be open to the public. If we want to collect or use the data containing personal information, we must comply with the personal information protection regulation like GDPR. The data published in public websites, like Kaggle, does not contain any personal information, such as user names and their location. Every person can use the Amazon review data achieved from Kaggle without getting any approval from the data publisher. The last criterion is the data size. Sentiment analysis is binary classification, whether the reviews present positive or negative. We need a certain amount of labeled data to train and test any classifiers like artificial neural networks. The Amazon review data contains 568,454 reviews for 74,258 products written by 256,059 users. The size of the selected data is enough to conduct a binary classification.

The raw data consists of 10 attributes, tabulated in Table 1. In sentiment analysis, an input feature is review text, and the class label is sentiment label, which is either positive or negative. The raw data has review texts, but there is no sentiment label. We need to convert the review score to a sentiment label. The other attributes are not useful for the classification. We perform data preprocessing to transform the raw data into an input feature and label.

**Table 1.** The attributes in the Amazon fine food review data

Attribute	Description
Row Id	Instance id
ProductId	Fine food product id on Amazon. Unique identifier for the product
UserId	User id who wrote the review. Unique identifier for the user
ProfileName	Profile name of the user
HelpfulnessNumerator	Number of users who found the review helpful
HelpfulnessDenominator	Number of users who indicated whether they found the review helpful or not
Score	Review score which is rating between 1 and 5 is best and 1 is worst
Time	Timestamp when review was saved
Summary	Brief summary of the review
Text	Text of the review

### 3. DATA CLEANING AND TEXT PRE-PROCESSING

#### 3.1. Data pre-processing

Most customer reviews, such as IMDB movie reviews and Yelp reviews, have a review score, which is a rating between 1 and 5. The Amazon review also follows this tradition. We perform the data preprocessing to convert the review score to the sentiment label. Figure 2 shows the procedure of data preprocessing. The first step is to convert the review score to a sentiment label. According to the published papers [2], the review score higher than 3 is defined as being positive, and lower than score 3 is assigned to negative. The reviews with a score of 3 represent the neutral, so they are not useful for the classification. We apply this definition to perform the first step. Next, we checked whether the data contains duplicated instances. Here, we define instances containing the same user id, profile name, time, and review text as duplicated data. Following the definition, we find that 37 000 duplicated positive and 25 000 duplicated negative reviews exist in the data. One of the duplicated reviews is discarded from the data to make the data containing unique reviews and their corresponding label. Finally, we delete the unnecessary attributes: Row Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, and Summary. The cleaned data contains 307,063 positive and 57,110 negative review texts and labels.

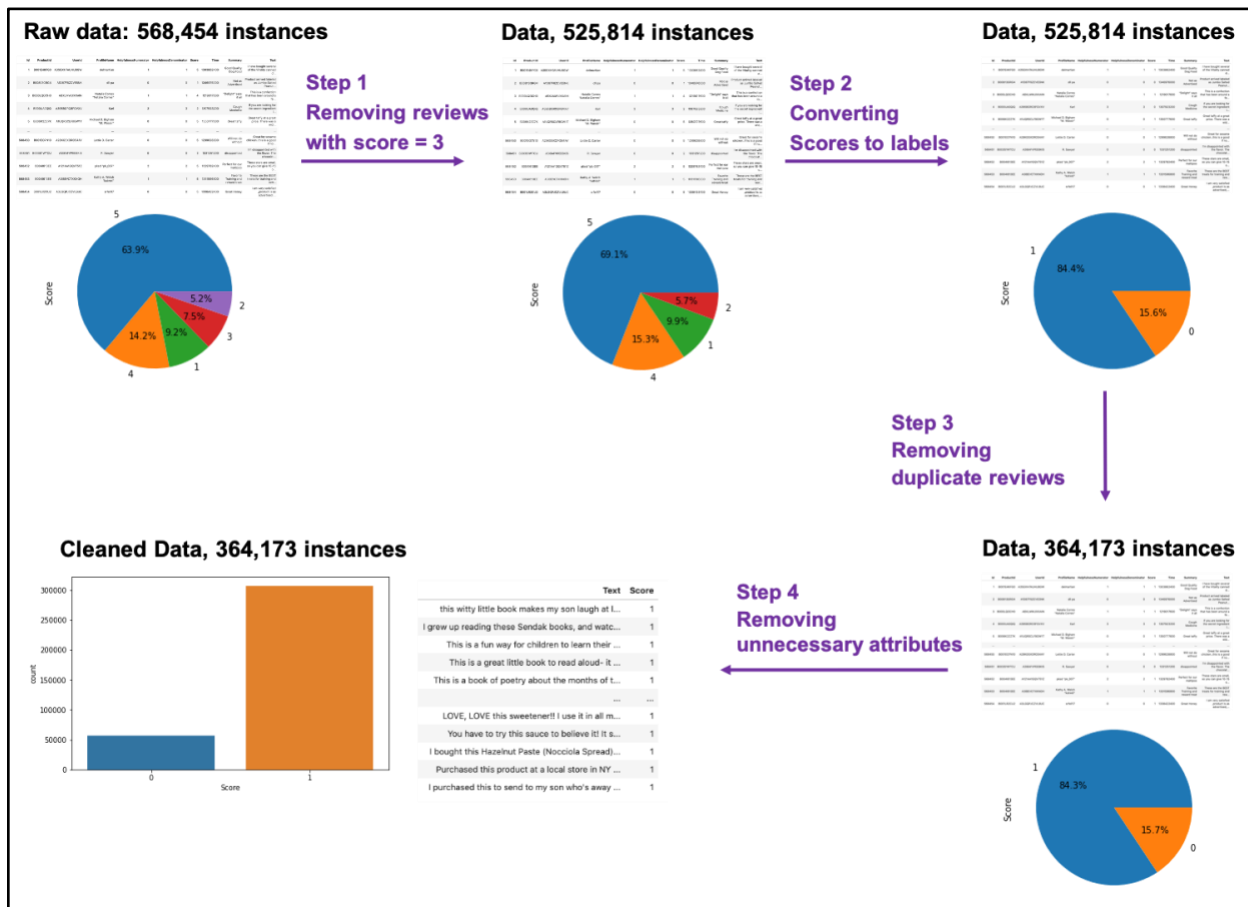


Figure 2. Data preprocessing procedure

#### 3.2 Text preprocessing

Every user can write a review on the Amazon website so that reviews may contain non-word text, such as URLs and HTML tags. The non-word texts are usually not used for any natural language processing because they do not contain any useful information for conducting NLP tasks. For example, URLs in a text show a location on the internet but do not provide any additional information about the sentiment. Moreover, the text data contains many stop words and contractions. The stop words are the most commonly used in a language. In other words, we can see stop words in every text. The stop words can not provide meaningful information for the classification tasks. To handle these texts, we perform text preprocessing, which consists of 4 steps. The first step is removing useless words or texts in the review. The useless words include website links (URLs), HTML tags, words with numeric digits, non-word characters, and stop words. The second step is decontracting. We select the 10 most popular contractions to convert the original forms (Table 2). The third step is converting to lowercase because the computer recognizes the different words that are the same spelling but different case. The last step is lemmatizing.

**Table 2.** The selected contractions for text preprocessing

Contraction	Original form
(1) won't, (2) can't, (3) n't, (4) 're, (5) 's, (6) 'd, (7) 'll, (8) 't (9) 've (10) 'm	(1) will not, (2) can not, (3) not, (4) are, (5) is, (6) would, (7) will, (8) not, (9) have, (10) am

The text preprocessing helps to reduce the input dimension by deleting non-useful words. Below is a representative example of the review in the data. This review is a mixed case, and contains HTML tag (<br>), stop words, words with numeric digits (250 mls), and stop words (I, of, and, to, and so on). The original review has 101 words. After text preprocessing, it contains only 56 words.

***Original Review (Before text preprocessing)***

Ginger has the somewhat contradictory quality of being both harsh and soothing, which makes for a great complex flavor. I think that flavor is captured well in this tea by Kili, and the sweet honey taste and mild lemon only serve to accentuate the ginger rather than steal the ginger's spotlight. I could drink this every day.<br /><br />In terms of quantity, the package calls for 250 mls of water, which fit a tea cup pretty perfectly. The granulated nature of the product means it is truly instant and dissolves perfectly, but it also seems to dull the flavor a bit.

***Review after text preprocessing***

ginger somewhat contradictory quality harsh soothing make great complex flavor think flavor captured well tea kili sweet honey taste mild lemon serve accentuate ginger rather steal ginger spotlight could drink every day term quantity package call ml water fit tea

cup pretty perfectly granulated nature product mean truly instant dissolve perfectly also seems dull flavor bit

Unfortunately, some reviews are converted to null values by text preprocessing. A representative example shows below. The example review contains all text in HTML format with a URL. The whole review text is removed by text preprocessing.

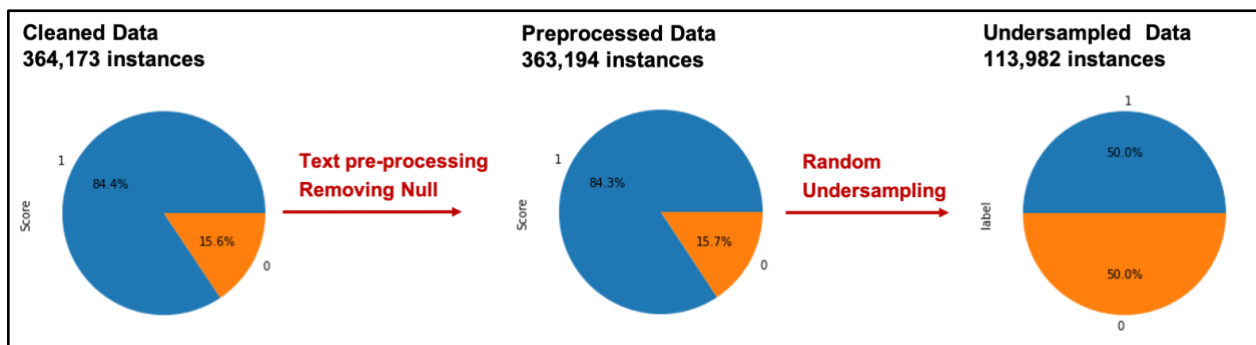
***Original Review (Before text preprocessing)***

'<a href="http://www.amazon.com/gp/product/B005K4Q1YA">Grove Square Cappuccino Cups, French Vanilla, Single Serve Cup for Keurig K-Cup Brewers, 24-Count</a> Very good. No need to add sugar.'

***Review after text preprocessing***

(Null)

After text preprocessing, we find 860 positive and 119 negative reviews converted to Null. The data contains 306,203 positive and 56,991 negative reviews. The data is too imbalanced to perform the classification. We do a random undersample to generate balanced training data for the classifier. Our final data is composed of an equal number of positive and negative reviews (Figure 3). Figure 3 shows the change in the balance ratio of the data during the text preprocessing.



**Figure 3.** The class balance rate of the data during the text preprocessing

## **4. PROPOSED APPROACH AND MODELS:**

### ***4.1. Benchmark: GloVe pre-trained word embedding + ANN***

An artificial neural network model with GloVe is selected as a benchmark model because it is the simplest deep learning model. The GloVe provides the pre-trained word vectors, which are available to download from <https://nlp.stanford.edu/projects/glove/>. We do not need to build and train any NLP models, such as the Continuous Bag of Words Model (CBOW) and Skip-Gram, for obtaining word embedding vectors. A multilayer perceptron with the GloVe pre-trained word vectors is easy to implement. This model may show the minimum performance (or accuracy) that

we can achieve from the deep learning-based models.

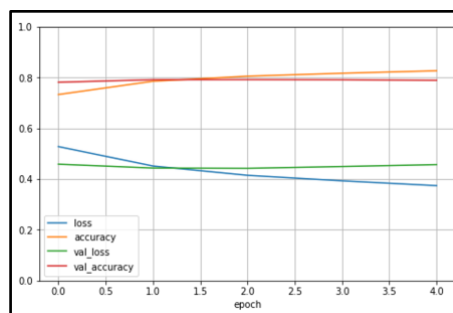
The model consists of 4 layers. The first layer performs word embedding using the GloVe pre-trained word vectors. The second layer works as an input layer, which flattens the embedded vectors. The second one is a hidden layer having 16 neurons. The last layer is the output. In this model, the RELU activation function is used for Hidden Layers and at Output Layer, Sigmoid activation function. The model summary shows in Figure 4(1). The dropout layer and the early stepping are applied to prevent the model's overfitting. Adam optimizer and binary cross-entropy loss function are applied to perform the backpropagation.

The benchmark model reaches convergence at 4 iterations (Figure 4(2)). It takes 166 seconds on the GPU mode on Google Colab. The confusion matrix is shown in Figure 4(3). The benchmark model achieved 79.26% accuracy, 79.54% precision, 79.26% recall, and 79.22% F1-score.

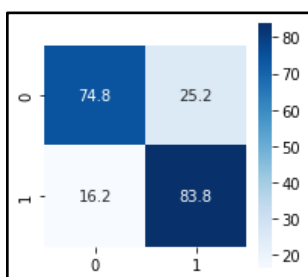
(1) Model summary

Model: "sequential"		
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1562, 300)	18097200
flatten (Flatten)	(None, 468600)	0
dense (Dense)	(None, 16)	7497616
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1)	17
Total params: 25,594,833		
Trainable params: 7,497,633		
Non-trainable params: 18,097,200		

(2) Learning curve



(3) Confusion matrix (Normalized)



**Figure 4.** Benchmark Model

#### 4.2. Word2vec and ANN (DNN)

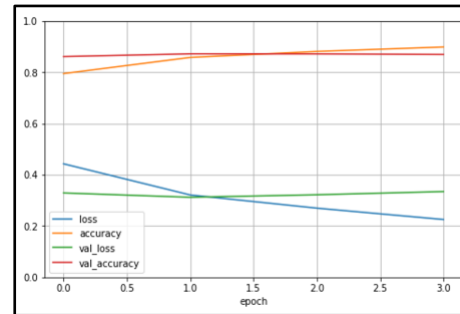
The first proposed model is a multilayer perceptron coupled with CBOW. Unlike the traditional classifiers, ANN does not require complicated feature engineering, such as feature selection by information gain and vector representation by Tf-idf. Furthermore, ANN achieves higher performance in the sentimental analysis than Naive Bayes and Support vector machines. There are many sophisticated deep learning models like RNN and pre-trained transformer-based models like BERT available. These models may achieve better performance than ANN models. However, their architecture is much more complex than ANN models because they are developed to perform various NLP tasks such as text generation, text summarization, and question answering. The computation cost, including computation power and running time, is

much more expensive. This project aims to construct a predictive model that tells whether any customer review expresses positive or negative sentiment. We need to produce a binary classifier, which is much simpler than most NLP tasks. When we consider the performance and computational cost, the ANN should be a good candidate model to conduct the project. The ANN, coupled with the Word2Vec, has 1 difference from the benchmark model. In this model, we perform CBOW using our training data. It can show how word embedding methods affect the performances of binary classification. The model architecture is the same as the Benchmark (Figure 5(1)). The model reaches convergence at 3 iterations (Figure 5(2)), which takes 1298 seconds on the GPU mode on Google Colab. The model training time is 7 times longer than the benchmark. However, this model achieved higher performances, 87.44% accuracy, 87.46% precision, 87.45% recall, and 87.44% F1-score. The confusion matrix is shown in Figure 5(3). The first proposed model, ANN+Word2Vec, improves performances by approximately 8%, but it requires more computation cost.

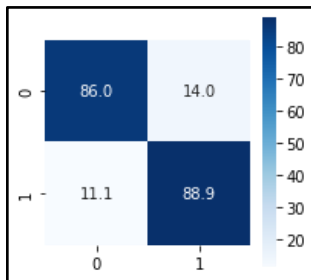
(1) Model summary

Model: "sequential"		
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1562, 300)	18097200
flatten (Flatten)	(None, 468600)	0
dense (Dense)	(None, 16)	7497616
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1)	17
Total params: 25,594,833		
Trainable params: 25,594,833		
Non-trainable params: 0		

(2) Learning curve



(3) Confusion matrix (Normalized)



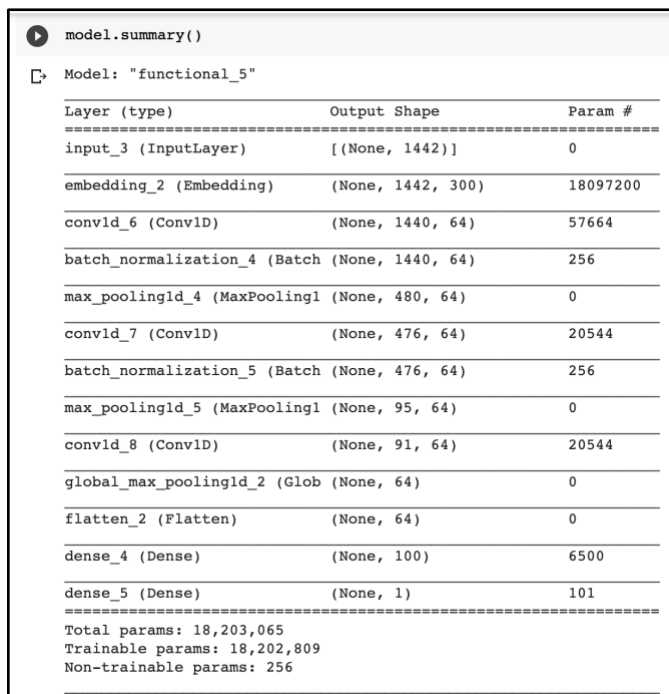
**Figure 5.** ANN with Word2Vec (CBOW) Model

### 4.3. Word2vec and CNN

The next model, Word2Vec- CNN, uses deep learning & word vector representation algorithms for sentiment analysis. We proposed this model because it achieves higher performance than the traditional classifiers when it comes to text classification. CNN is also easier to train and can achieve higher accuracy compared to simple deep learning models such as DNN (ANN).

Among the existing studies using deep learning to classify texts, CNN takes advantage of the convolutional filters that automatically learn features suitable for the given task [3]. For example, for sentiment classification, the convolutional filters of CNN may capture inherent syntactic and

semantic features of sentimental expressions [3]. It has been shown that a single convolutional layer, a combination of convolutional filters, may achieve comparable performance even without any special hyperparameter adjustment [3]. The proposed model will help to classify better text classification of small text sentences, such as reviews. CNN is effective for text classification and can be extended for this use with help of word embedding. The word embedding used in this case was Word2vec, which can convert text to vectors or word embedding which can be then fed into CNN model. We performed CBOW using our training data. It can show how different word embedding algorithms affect the performances of binary classification. CNN architecture has the 1D convolutional and pooling operation. In NLP tasks, when CNN is applied to text instead of images, one dimensional array representation of the text is required. Pooling operation, in this case max pooling, is also used to combine the vector results from different convolution windows into a single 1-dimensional vector and this vector will capture the most relevant features of the sentence. The CNN-Word2vec model reaches convergence at 2 epochs, giving us an accuracy score of 86.26%, precision, 84.89%, recall, 87.81%, and f1 score, 86.33%. Figure 6 below is the model architecture used for the CNN model.



```
model.summary()
```

Model: "functional\_5"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 1442)]	0
embedding_2 (Embedding)	(None, 1442, 300)	18097200
conv1d_6 (Conv1D)	(None, 1440, 64)	57664
batch_normalization_4 (Batch Normalization)	(None, 1440, 64)	256
max_pooling1d_4 (MaxPooling1D)	(None, 480, 64)	0
conv1d_7 (Conv1D)	(None, 476, 64)	20544
batch_normalization_5 (Batch Normalization)	(None, 476, 64)	256
max_pooling1d_5 (MaxPooling1D)	(None, 95, 64)	0
conv1d_8 (Conv1D)	(None, 91, 64)	20544
global_max_pooling1d_2 (GlobalMaxPooling1D)	(None, 64)	0
flatten_2 (Flatten)	(None, 64)	0
dense_4 (Dense)	(None, 100)	6500
dense_5 (Dense)	(None, 1)	101

Total params: 18,203,065  
Trainable params: 18,202,809  
Non-trainable params: 256

**Figure 6.** CNN Model Summary using Word2Vec

#### 4.4. Word2vec and LSTM (RNN)

The fourth model is our word2vec lstm model. The data is run through a word2vec algorithm in order to vectorize the words in our Amazon reviews. Which is then passed through the LSTM. Each word is vectorized using the Word2Vec model and fed into Long Short-Term Memory (LSTM). LSTM, derived from RNNs, works by processing data passing on information as it moves forward, within LSTM cells. Unlike RNNs, LSTM works by only remembering the useful information and then forgetting the rest, which solves the long term dependency issue that often



occurs with RNNs. This chain like structure interacts through the cells with the ability to remove or add information by gates. The gates are a way for the information to be let through or not. The first step is to divide which information to be thrown away, then decide which information is going to be kept and finally then you update the old cell into a new cell, and the values are updated. Finally, it is decided what the output is going to be based on the cell state. The overall idea is to let each step of the neural network decide and pick which information will be looked at. The benefits of using Word2Vec & LSTM (RNN) are: LSTMs work really well with traditional NLP tasks; LSTM was implemented to solve the vanishing gradient problem in RNNs; The complexity to update each weight is reduced  $O(1)$  with LSTMs. One drawback is LSTM requires a lot of resources and time to get trained and ready. “Because RNN is prone to gradient explode or gradient vanish during training, it is difficult to learn long-term dependencies.”[4] The summary of our LSTM model is below.

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 150, 256)	7651584
lstm (LSTM)	(None, 128)	197120
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8256
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65
Total params: 7,857,025		
Trainable params: 205,441		
Non-trainable params: 7,651,584		

**Figure 7.** LSTM model Summary

#### ***4.5. Benchmark: Logistic Regression with Tf-Idf (Term Frequency-Inverse Document Frequency)***

Logistic Regression with Tf-Idf (Term Frequency-Inverse Document Frequency) is another baseline model using machine learning. We chose this model to compare deep learning model performance with a machine learning model. Logistic regression is known for easily understandable mechanisms and cheaper computational-cost. Tf-idf is a vectorizer used to tune logistic regression models. It stands for term frequency–inverse document frequency. Tf-idf vectorizer words by taking into account the frequency of a word in a given document and the frequency between documents.

Formula:  $\text{tf-idf}(w, d) = \text{bow}(w, d) * \log(N / \# \text{ documents in which word } w \text{ appears})$

Figure 8 shows the performance result- accuracy, F1 score, confusion matrix, recall, precision, and ROC AUC, of Logistic Regression with Tf-Idf Performance Result on Google Colab.

```

[18] #Baseline - LogReg(Tf-Idf)

[19] def prediction(model, X_train, y_train, X_valid, y_valid):
    model.fit(X_train, y_train)
    pred = model.predict(X_valid)
    acc = accuracy_score(y_valid, pred)
    f1 = f1_score(y_valid, pred)
    conf = confusion_matrix(y_valid, pred)
    precision = precision_score(y_valid, pred)
    recall = recall_score(y_valid, pred)
    roc_auc = roc_auc_score(y_valid, pred)
    joblib.dump(model, f"model_acc_{acc:.5f}.pkl")
    return model, acc, f1, conf, recall, precision, roc_auc

[20] transformer = TfidfVectorizer(stop_words='english', ngram_range=(1, 3),
                                lowercase=True, max_features=100000)
X = transformer.fit_transform(train_val['sentences'])
y = train_val.labels

[21] X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,
                                                         random_state=42, stratify=y)
model = LogisticRegression(C=1, random_state=42, n_jobs=-1)
fit_model, acc, f1, conf, recall, precision, roc_auc = prediction(model, X_train, y_train, X_valid, y_valid)

[22] print(f"Accuracy: {acc:.5f}")
print(f"F1_Score: {f1:.5f}")
print(f"Confusion Matrix: {conf}")
print(f"Recall: {recall:.5f}")
print(f"Precision: {precision:.5f}")
print(f"ROC_AUC: {roc_auc:.5f}")

Accuracy: 0.88552
F1_Score: 0.88619
Confusion Matrix: [[8061 1081]
 [1017 8168]]
Recall: 0.88928
Precision: 0.88312
ROC_AUC: 0.88552

```

**Figure 8.** Logistic Regression with Tf-Idf Performance Result on Google Colab

## 5. EVALUATION METRICS

Evaluation metrics are used to qualify how well our deep learning models work. In order for us to determine a model that provides the best outcomes, we used the following models:

### *Performance*

1. Accuracy
2. ROC AUC score

### *Computational cost*

1. Average training speed per epoch in seconds (for Deep Learning models only)
2. Iteration to reach the deep learning model convergence

The receiver operating characteristic chart, ROC, is a graph that shows the plot of performance of the classification models. The two parameters that are plotted are the true positive rates and the False positive rates. A true positive score tells us what proportion of the positive class we got correctly and a false positive rate tells us the proportion of the negative class we got incorrectly.

The area under the curve, AUC, score works by measuring the two-dimensional area under the ROC curve, integral from (0,0) to (1,0). The AUC score shows the probability that the model ranks more positive or negative. It measures how well predictions are ranked. Therefore a score of 1 then the model is able to perfectly distinguish between the positive and negative class. However, if the score is - then the model predicts all negatives as positives and vice versa. When we have a score of 0.5 then our model was not able to predict very well. Generally we are looking for a higher AUC value.

## **6. GENERAL PROCEDURE**

The general procedure that we had taken before implementing our deep learning model, in this case: ANN, CNN, and LSTM (RNN) models include following:

1. Split pre-processed data into train and test datasets (done during preprocessing)
2. Load both these datasets
3. Combine both training and testing datasets into one corpus of text data and then converting the text into vector
4. Tokenize corpus data, which allows us to vectorize a text corpus, by turning each text into either a sequence of integers in this case.
5. Apply zero padding to corpus data so the length of all the sequences are the same. Basically, ensure that the inputs are of the same size and shape which is required when training neural networks.
6. Create and train word2vec model, using gensim package, on our corpus
7. Constructing a dictionary to extract the word vectors from
8. Create embedding matrix for corpus using word2vec
9. Finally, feed word2vec word embedding matrix as input into deep learning model

## **7. MODEL RESULTS AND COMPARISON**

The table 3 below shows a comparison of the values required to calculate the computation costs associated with training each deep learning model.

**Table 3**

Computational Cost	Benchmark model	Word2Vec embedding + ANN	Word2Vec embedding + CNN	Word2Vec embedding + LSTM (RNN)	Logistic Regression + Tf-Idf
Training speed (second per epoch)	41.5	429.7	154.5	690.9	28 (total sec)
Number of epochs to reach convergence	4	3	2	5	NA

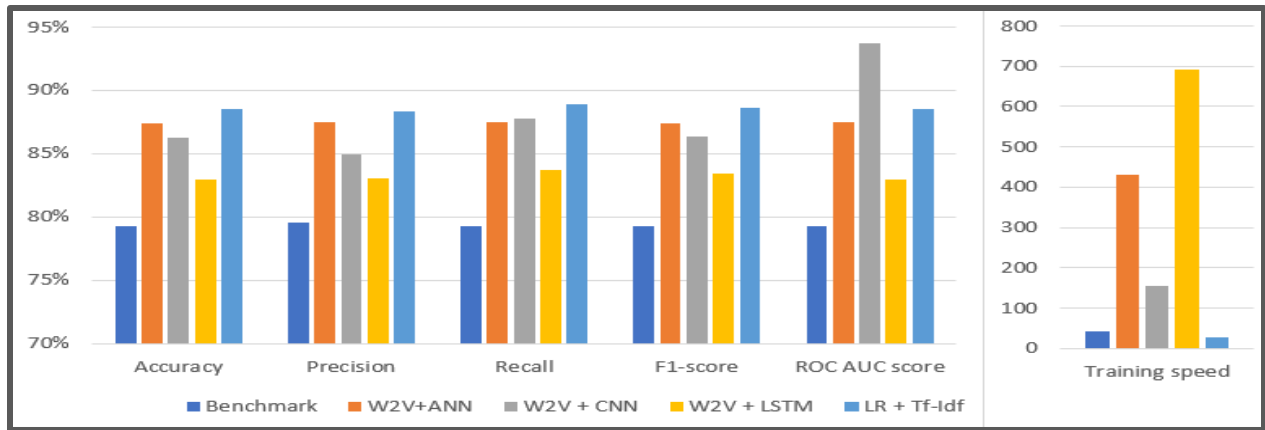
We decided to calculate computation cost for each of our model using:

1. Average training speed per epoch in seconds and
2. Number of epochs to reach model convergence

We chose epochs because the number of epochs (iterations) to reach convergence tells us how long the model training takes. Even though each iteration is completed for a short amount of time, the *whole* model may need to go through hundreds, or even more, iterations total, and that can take a long time. For example, although the benchmark model training speed on average was 41.5 seconds per epoch, we don't know for sure how many iterations this benchmark model had gone through during the whole training process. So, in order to evaluate the whole model training time, we chose epochs.

Therefore, to calculate the whole training time of the model, we will: Multiply the (training speed per epoch) by the (total number of epochs to reach the model convergence). We can do this for each model and determine which one of them is the most computationally expensive to train. We also know that the higher training speed (sec per epoch), the more computationally expensive model is. According to our results, the LSTM is the most computationally expensive model out of all. It seems like the complexity of a model influences training speed then. Well, we can't really say that because ANN is the least complex DL model yet takes more time to train compared to the CNN model, which is more complex. One possible reason for this difference could be that CNN reduces size of inputs (dimensionality reduction) and thus much quicker compared to other complex models, such as LSTM.

Figure 9 depicts our performance metrics for each of our models: Accuracy, ROC AUC score, and other metrics.



**Figure 9.** Performance metrics for each of our models

From the get-go, we can see that among the deep learning models that used W2vec embeddings as inputs, ANN scored the highest, followed by CNN, and then the LSTM model. All three scored higher than the benchmark model (which used GloVe pre-trained word embeddings instead, followed by ANN). Only difference between the implementation of the benchmark model and W2vec + ANN model is that the trainable parameter is set to false in the benchmark as we don't want to change our pre-trained matrix. Logistic regression with Term Frequency-Inverse Document Frequency is another baseline model to compare deep neural network algorithms with simple machine learning models. Logistic Regression + Tf-Idf model scored the highest accuracy compared to the rest of the deep learning models that used W2vec possibly because Tf-idf vectorizes words by taking into account the frequency of a word in a given document and the frequency between documents. We also see that our ROC AUC scores for all our models increased from the baseline score, which is good and indicates that our classes have been well separated. ROC AUC score is a useful metric in cases of imbalanced dataset, which we did have in the beginning. However, since we had corrected for our imbalanced dataset during pre-processing, we can safely use accuracy as a reliable metric for measuring our model performance.

## 8. REFERENCE

- [1] P. by D. Coppola and N. 27, "U.S. e-commerce market size 2016-2023," *Statista*, 27-Nov-2020. [Online]. Available: <https://www.statista.com/statistics/272391/us-retail-e-commerce-sales-forecast/>. [Accessed: 09-Dec-2020].
- [2] R. Moraes, J. F. Valiati, and W. P. G. Neto, "Document-level sentiment classification: An empirical comparison between SVM and ANN," *Expert Systems with Applications*, vol. 40, no. 2, pp. 621–633, 2013.
- [3] H. Kim and Y.-S. Jeong, "Sentiment Classification Using Convolutional Neural Networks," *Applied Sciences*, vol. 9, no. 11, p. 2347, Jun. 2019 [Online]. Available: <http://dx.doi.org/10.3390/app9112347>
- [4] C.Wang. D.Han. Q.Liu. and S.Luo, "A Deep Learning Approach for Credit Scoring of Peer-to-Peer Lending Using Attention Mechanism LSTM," *IEEE Access*, Jan. 2019 [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8579130>

## 9. DATA & MODEL SOURCE & TASK DISTRIBUTION

- PPT slides link: <https://docs.google.com/presentation/d/16dlZWz90hk9hQQDGL-KlvRXjrbz86MiZut-8W2N0JI0/edit?usp=sharing>
- Youtube link: <https://www.youtube.com/watch?v=gqinoEsgRIU>
- Raw data source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>
- Source code: <https://drive.google.com/drive/folders/0ALp7lVBX9JT0Uk9PVA>

	Hye In	Afra	Kimberly	Ginny (Jeehee)
Raw Data	Reviews.csv	-	-	-
Stopwords	english*	-	-	-
Glove Pretrain word vector	glove.6B.300d.txt.zip **  (For ANN_GloVe)	-	-	-
Source Code	ANN_GloVe_Hyein Nam.ipynb	CNN_W2V_AfraBijli .ipynb	Word2VecLSTM_KS egura.ipynb	LogisticRegression_T f_Idf4_Choi.ipynb
	ANN_W2V_HyeinN am.ipynb			
	data_preprocessing_ HyeinNam.ipynb			
Train & Test Data	train_set3.csv			train_test_set3.csv
	test_set3.csv			
Link for Large Size Files	-	-	-	-

\* The stopword list is used for data preprocessing.

\*\* The original file, GloVe.6B.300d.txt, is 990MB, so we compressed it for uploading to the shared drive.