**CS3244 Final Report Group 9**

Aung Naing Tun, Mavis Neo, Nicholas Jimmy Alden,
Nixon Widjaja, So Wai Yein Ethan, Sum Hung Yee

## 1. Project Background

### 1.1 Problem Motivation

The study of the interaction between human language and computers is known as NLP or Natural Language Processing. One area in NLP that we will explore is sentiment analysis, which involves determining whether a statement is positive or negative.

Analyzing consumer sentiment is important for companies in general, as they wish to learn more about the demand for their product. Take Apple, for instance, who needs to know what its users think about the iPhone. However, with no rating system for its products, a good way to learn more about the demand for its products is by analyzing reviews made by users of the product. However, manually reading through reviews on various platforms such as social media would be time-consuming and costly, and thus these processes should be automated. By processing large amounts of data and classifying comments, Apple can gain more information about its position in the market and adapt more readily to changes in the market. Therefore, in this project, we will explore sentiment analysis, which can be beneficial for companies to understand the market and improve their products and for consumers to make informed decisions when purchasing goods. We will investigate and optimize various machine learning models, including the State-of-the-Art (SOTA) model, to perform sentiment analysis and compare their performance.

### 1.2 Application outline

Amazon is one of the largest e-commerce platforms in the world, with millions of products and reviews. As a seller or a buyer, understanding the sentiment of those reviews can be critical to making informed decisions. This is where our machine learning application comes in. We aim to make a machine learning application that uses natural language processing (NLP) algorithms, i.e., sentiment analysis to analyze Amazon reviews and classify them as positive or negative based on their sentiment. This application will provide a classification summary of the overall sentiment for a product, allowing users to quickly gauge customer sentiment and make informed decisions. Our user demographics for this application are Amazon sellers, buyers, and researchers who need to analyze large volumes of Amazon reviews. From this project, we will determine from various machine learning models which one can produce the highest accuracy in classifying whether a product has a positive or negative review.

### 1.3 Data Information

Link: https://www.kaggle.com/datasets/bittlingmayer/amazonreviews

| Columns | Data Type | Details |
|---|---|---|
| Text | String | Reviews on Amazon products |
| Label | Integer | 0: for reviews with negative sentiments<br>1: for reviews with positive sentiments |

Since the reviews are taken straight from the Amazon site, it still contains a lot of noise from irrelevant tokens (e.g. URLs, emojis, etc.) and therefore requires cleaning, before being passed to the feature extraction process.

The label distribution seems to be pretty balanced, so there should be no need for any oversampling. From the 3.6 million entries within the training set, 1.8 million are labeled as positive and 1.8 million are labeled as negative. The testing set contains 400,000 entries and the positive and negative labels have exactly 200,000 entries each.

For the purpose of showing the viability of our model, we will feed only the first 100,000 entries of the training set to the model and test with the first 25,000 entries of the testing set. The distribution remains more or less similar for both the training and testing set which will be explained in the following section.

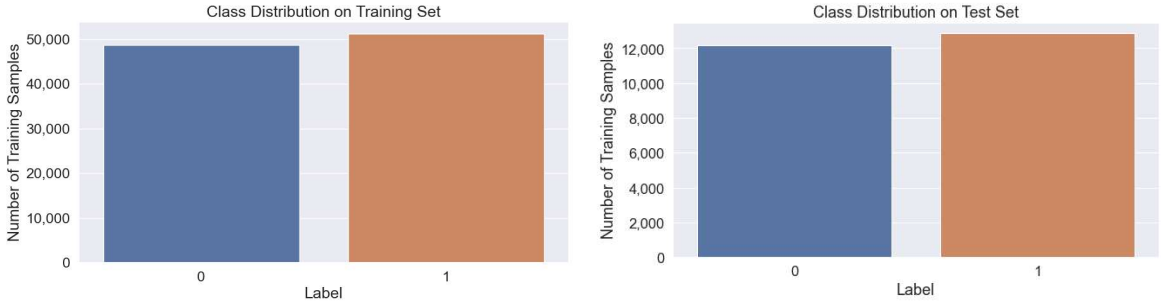
## 2. Analytical Methods & Justification

### 2.1 Exploratory Data Analysis (EDA)

In order to gain more overall insights into the whole dataset, we have applied EDA on the dataset. EDA helps in understanding the data we are working with. By visualizing and analyzing the data, we can identify the distribution of the profile of the dataset and get some analysis of the overall dataset.

*Class Distribution*

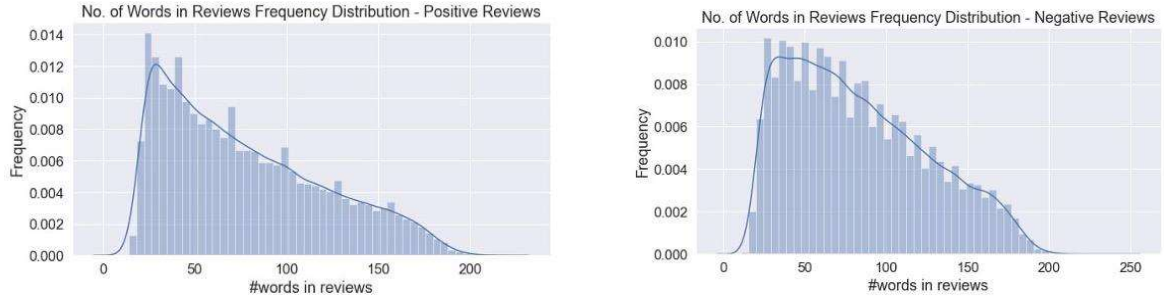
Class distribution is an important aspect to consider during EDA in a machine learning project as it helps to identify the proportion of each class in the dataset and prevents biases towards any particular class.




Both our training and test dataset is fairly balanced with approximately equal number of labels in the datasets. This will ensure that our models are not biased towards any of the labels and hence no additional techniques such as oversampling is needed to handle imbalanced classes.

*No. of Words in Reviews Frequency Distribution*

The number of words in reviews can help to identify any potential patterns which might assist in the sentimental analysis.



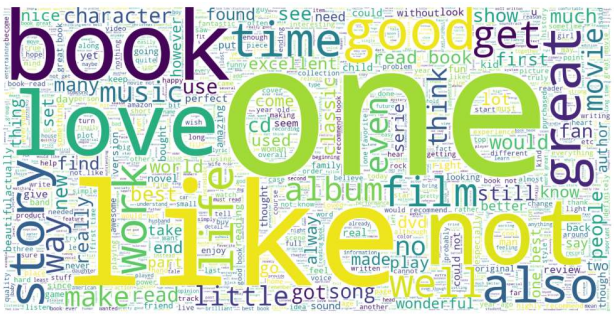

Both Positive and negative reviews have a similar range of number of words per review where the bulk of reviews fall in the range of 0 to 200 words and a similar distribution where number of words for both mostly fall between 25 to 75 words per review.

*Wordcloud*

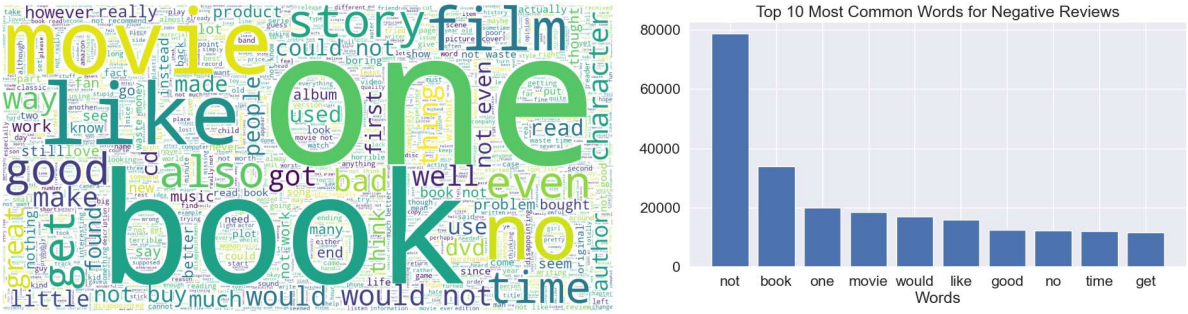
An additional EDA we can perform is to generate a world cloud based on reviews from the training dataset. Word cloud creates visualization based on frequency of occurrence of word in review and this could give us an idea of the most common words used in the reviews of each label.

For Wordcloud EDA, we had to do Preprocessing on the dataset first which will be explained later in Section 2.2 Data Pre-processing. Additionally, we did the Wordcloud on only the Training Dataset which were separated into Positive and Negative Reviews accordingly.

Positive Reviews




Negative Reviews




The word cloud for both the positive and negative reviews under the training dataset share many similarities. Words like "book", "one" and "read " are the most common words in both reviews and this is not unexpected given the context of this dataset is reviews from amazon.

### 2.2 Data Preprocessing & Baseline Models

In order to produce the word cloud as shown previously, text cleaning needs to be performed beforehand. We remove URLs, remove hashtags, convert all words to lowercase, spell out contractions, remove punctuations, remove numbers, and remove special characters.

Additionally, word vectorization is a methodology in NLP to map words or phrases from vocabulary to a corresponding vector of real numbers which is used to find word predictions, word similarities/semantics. To determine the more suitable vectorizer, we performed an initial experiment by using both TF-IDF Vectorizer and CountVectorizer before running them with our baseline models. Comparing the accuracy from our proposal, we can conclude that TF-IDF outperforms CountVectorizer and will proceed with TF-IDF for the rest of our remaining models.

| | TF-IDF Vectorizer | CountVectorizer |
|---|---|---|
| Optimized kNN | 0.750 | 0.681 |
| Optimized DT | 0.719 | 0.712 |

## 3. Models Building & Exploration

### 3.1 Models Used & Methodology

For the models, we have chosen to use the following models:

- Baseline Models: K-Nearest Neighbors and Decision Trees
- Improved Models: Naive Bayes and Support Vector Machine
- SOTA Model: CNN-Bidirectional LSTM Model

We will be conducting exploration on the Improved Models and SOTA Model while the Baseline Models have been explored in our proposal earlier on.

When building the model, we would find the most optimal combination of hyperparameters by using GridSearchCV for the improved models and keras tuner for the sota model. We then fit the optimizer to the training data using grid_search.fit, which searches through the hyperparameters specified in parameters to find the best model. Finally, we use the best hyperparameters to make predictions on the test data and calculate the accuracy of the predictions using accuracy_score.

To evaluate the performance of the models, we would use accuracy as the main evaluation metric since our dataset is well balanced and we want to maximize the number of correct predictions.

Confusion matrix and classification matrix is also included to observe model performance for the individual classes.

### 3.2 Evaluation of Models Performance

Improved Model 1 -Naive Bayes

*Naive Bayes Model Methodology*

For Naive Bayes, we have chosen a 5-fold cross validation while searching the following parameters:

```
{'alpha': [0.1, 0.5, 1],  'fit_prior': [True, False] }
```

The hyperparameters 'alpha' and 'fit_prior' are specific to the Naive Bayes algorithm and have a significant impact on the model's performance.

'alpha': This hyperparameter controls the smoothing of the estimated probabilities. Smoothing helps to avoid overfitting when the model encounters new data. The alpha value determines the strength of the smoothing, with smaller values indicating stronger smoothing. In this case, we are considering the values of 0.1, 0.5, and 1 for alpha, which are commonly used values for Naive Bayes models.

'fit_prior': This hyperparameter determines whether or not to learn class prior probabilities from the data. If 'fit_prior' is set to True, the class prior probabilities will be learned from the training data. If set to False, the model assumes a uniform prior for all classes.

The range of hyperparameters chosen for this model is a reasonable starting point for tuning the Naive Bayes algorithm. The specific values were likely chosen based on previous experiences with the algorithm and some domain research about the data being used. By trying out different combinations of hyperparameters, we can determine which values work best for the specific problem at hand

By using GridSearchCV, the best hyperparameters and results are as follows:

*Best hyperparameters: {'alpha': 1, 'fit_prior': False}*

```
Naive Bayes Accuracy: 0.84472
Naive Bayes Confusion Matrix:
 [[10313  1844]
 [ 2038 10805]]
Naive Bayes Classification Matrix:
              precision    recall  f1-score   support

           0       0.83      0.85      0.84     12157
           1       0.85      0.84      0.85     12843

    accuracy                           0.84     25000
   macro avg       0.84      0.84      0.84     25000
weighted avg       0.84      0.84      0.84     25000
```

Improved Model 2 - Support Vector Machine (SVM)

*Support Vector Machine Methodology*

For SVM, we have chosen a 5-fold cross validation while searching the following parameters:

```
{'kernel': [linear],  'C': [0.1, 1, 10] }
```

We first define an SVM model and the hyperparameters to tune using GridSearchCV. We also specify a 5-fold cross-validation using cv=5.

We have chosen LinearSVC() as it is the best kernel hyperparameter from our proposal.

The hyperparameter C is one of the most important hyperparameters for the SVM model. The C hyperparameter controls the trade-off between the margin and the misclassification error of the training data. A smaller C value will create a wider margin, allowing more margin violations but potentially better generalization to new data. A larger C value will create a narrower margin, resulting in fewer margin violations but potentially overfitting to the training data.

By using the values [0.1, 1, 10] for C, the GridSearchCV optimizer will try all possible combinations of these hyperparameters, allowing us to find the best combination that maximizes the performance of the SVM model on the training data. The specific value chosen for C depends on the dataset and the problem at hand, and may need to be adjusted based on the results of the optimizer.

By using GridSearchCV, the best hyperparameters and results are as follows:

*{'C': 0.1'}*

```
SVC Accuracy: 0.88872
SVC Confusion Matrix:
 [[10676  1481]
 [ 1301 11542]]
SVC Classification Matrix:
              precision    recall  f1-score   support

           0       0.89      0.88      0.88     12157
           1       0.89      0.90      0.89     12843

    accuracy                           0.89     25000
   macro avg       0.89      0.89      0.89     25000
weighted avg       0.89      0.89      0.89     25000
```

SOTA Model - CNN-Bidirectional LSTM Model

*CNN-Bidirectional LSTM Model Methodology*

This neural network is inspired by a similar architecture as outlined by Kumar J et al (J et al., 2021).

The LSTM model is a specialized kind of Recurrent Neural Network. In particular it has a "forget" gate, which allows the neural network to discard unimportant components of sentences. Like other RNNs, a LSTM model deals with sequential data, such as text. In particular, we use Bidirectional LSTM layers, which will take into account forward and backward sequences, which enhances the learning of the network. Additionally, a 1-D CNN layer is positioned before the Bidirectional LSTM layer in order to extract local features of text. As a result of the model's adaptability, we can allow the neural network to learn what words are unimportant, which allows us to not eliminate stopwords, which are previously removed in the other models. This has an important advantage over traditional methods. This model will take into account the position of words within a sentence. For instance "not bad but good" is positive, while "not good but bad" is negative. Furthermore, taking into account text as a sequence allows the neural network to understand context to a better extent than just simply detecting the count or presence of certain words/tokens. For our model, we have chosen the Keras Tuner while searching the following parameters:

```
dims = hp.Int('dims', min_value=48, max_value=96, step=16)
kernel_size = hp.Int('kernel_size', min_value=3, max_value=6, step=1)
filters = hp.Int('filters', min_value=24, max_value=96, step=16)
pool_size = hp.Int('pool_size', min_value=2, max_value=8, step=2)
learn_rate = hp.Choice('learn_rate', values=[1e-4, 5e-5])
```

“dims” represents the dimension of the embedding vector within the Embedding layer, “kernel_size” represents the size of the kernel in the 1-dimensional Convolutional-1D layer, “filters” represents the number of kernels, and “pool_size” is the pool size of the MaxPooling1D layer.

This is a sequential neural network model with CNN as well as Bidirectional LSTM layers. The embedding layer first converts each integer token into an n-dimensional array, where n is a hyperparameter “dims”. The greater n is, the better the model will be able to capture relations between words. However, this is only effective to a certain extent, and an n too great may result in overfitting.

As outlined in the paper by Gopalakrishnan et al and Kumar J et al, the next layer is a CNN layer which will look at local features about each token, and extract this information. This is followed by a MaxPooling1D layer. This helps to account for the idea that some words, when placed in proximity with other words, change the meaning of the phrase. e.g. "not good". The kernel size of the CNN layer, learning rate, dimension of the embedding vector, the number of convolutional filters and the pool size of the 1-D max-pooling layer are also other hyperparameters to be tuned.

The bi-directional LSTM layers that follow treat the series of tokens (a review) as sequences, which helps the neural network understand the underlying context of the review, rather than simply capturing the presence of words. The presence of multiple LSTM layers helps to abstract out possible nested sequences, which aids learning even further.

By using the CNN-Bidirectional LSTM Model, the best hyperparameters and results are as follows:

*{'dims': 96, 'kernel_size': 3, 'filters': 72, 'pool_size': 2, 'learn_rate': 0.0001, 'tuner/epochs': 16, 'tuner/initial_epoch': 0, 'tuner/bracket': 0, 'tuner/round': 0}*

```
SOTA Accuracy: 0.8915200233459473
SOTA Normalised Confusion Matrix:
[[0.88718537 0.10435054]
 [0.11281463 0.89564946]]
SOTA Classification Matrix
              precision    recall  f1-score   support

           0       0.89      0.89      0.89     12157
           1       0.90      0.89      0.89     12843

    accuracy                           0.89     25000
   macro avg       0.89      0.89      0.89     25000
weighted avg       0.89      0.89      0.89     25000
```

### 3.3 Insights/Findings & Conclusion

| Model | Accuracy |
|---|---|
| Naive Bayes | 0.845 |
| Support Vector Machine | 0.889 |
| CNN-Bidirectional LSTM Model | 0.892 |

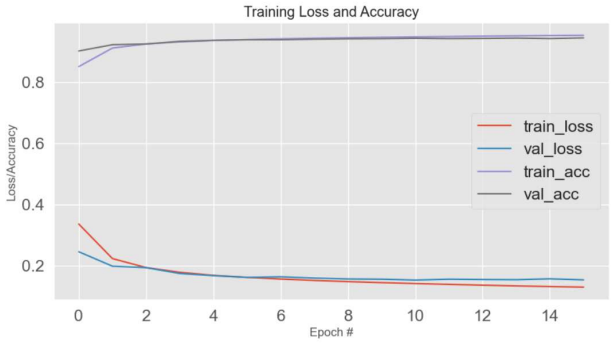
We find that CNN-Bidirectional LSTM is the best model since it has the highest accuracy and an f1-score better than the next best model, the linear SVM.

### 3.4 Extension of the Final Model: CNN-Bidirectional LSTM

Having selected CNN-Bidirectional LSTM as our final model, we realize that the number of datapoints, at 100,000, is insufficient for optimal results. We then trained it on the full dataset with 3.6 million data points to better evaluate the performance of CNN-Bidirectional LSTM model.

The best hyperparameters and results, found through Keras-Tuner, are as follows:

{'dims': 112, 'kernel_size': 3, 'filters': 128, 'pool_size': 8, 'learn_rate': 1e-05}




```
In [29]: 1 model.evaluate(X_test, y_test)
         12500/12500 [==============================] - 121s 10ms/step - loss: 0.1549 - accuracy: 0.9440
Out[29]: [0.15485890209674835, 0.9439975023269653]
```

```
In [36]: 1 print(confusion_matrix(y_test, np.round(y_prediction), normalize= 'pred')
         [[0.948684   0.06059212]
          [0.051316   0.93940788]]
```

This time, the parameters searched were:

```
dims = hp.Int('dims', min_value=64, max_value=120, step=16)
kernel_size = hp.Int('kernel_size', min_value=3, max_value=6, step=1)
filters = hp.Int('filters', min_value=16, max_value=128, step=16)
pool_size = hp.Int('pool_size', min_value=2, max_value=8, step=2)
learn_rate = hp.Choice('learn_rate', values=[1e-5, 1e-6, 1e-7])
```

The CNN-Bidirectional LSTM achieved a great accuracy of 94.4% when trained on the full dataset. The confusion matrix that follows shows a mostly equal distribution of False Positives and False Negatives. The f1-score for this is evaluated to be 2 / ( (0.949 + 0.061) / 0.949 + (0.949 + 0.051) / 0.949) = 0.944.

### 3.5 Conclusion & Future Direction

The results obtained from training the CNN-Bidirectional LSTM on the full dataset demonstrate the model's potential to achieve high accuracy in NLP, especially sentiment analysis. However, there are still areas of improvement and avenues for future research to consider.

The model's performance could be further optimized by further fine-tuning its hyperparameters. We have experimented with various combinations of learning rates, dimensions of the embedding, and other model settings to determine the most effective configuration for this particular task. However, more work could be done on finetuning the batch-sizes. Furthermore, we note that other more-refined architectures exist, such as the Transformer, and in particular, BERT. Additionally, the performance of the model could be enhanced by using more advanced techniques such as data augmentation or transfer learning. While the model demonstrated strong performance when trained on the full dataset, it is important to test its performance on a larger and more diverse dataset to validate its effectiveness in real-world scenarios. This could involve using additional data sources to expand the size and scope of the training set, or evaluating the model's performance on data from different domains or contexts.

Finally, to fully utilize the potential of the SOTA model, it is essential to have proper infrastructure and resources such as high-end GPUs and sufficient processing power. Future work could focus on exploring more efficient and scalable methods for training and deploying the model, such as distributed computing or cloud-based solutions.

In summary, while the CNN-Bidirectional LSTM achieved impressive results on the full dataset, there are still opportunities for optimization and further research. By exploring these avenues and leveraging advanced technologies, we can unlock the full potential of this SOTA model in sentiment analysis.

## 4. References


1. Pedregosa et al.: *'Scikit-learn: Machine Learning in Python'*. JMLR 12. pp. 2825-2830 (2011)

2. Harris, C.R., Millman, K.J., van der Walt, S.J. et al.: *'Array programming with NumPy'*. Nature 585. pp. 357–362 (2020).

3. McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 56-61).

4. The pandas development team. (2020, February). pandas-dev/pandas: Pandas 1.5.3. Zenodo.

5. Bird, S., Loper, E., Klein, E.: '*Natural Language Processing with Python'*. O'Reilly Media Inc (2009).

6. Gopalakrishnan, K., & Salem, F. M. (2020). Sentiment Analysis Using Simplified Long Short-term Memory Recurrent Neural Networks. CoRR, abs/2005.03993. Retrieved from https://arxiv.org/abs/2005.03993

7. J, A.K., Trueman, T.E. & Cambria, E. A Convolutional Stacked Bidirectional LSTM with a Multiplicative Attention Mechanism for Aspect Category and Sentiment Detection. Cogn Comput 13, 1423–1432 (2021)

8. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, ... Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Retrieved from https://www.tensorflow.org/

9. Chollet, F., & Others. (2015). Keras. Retrieved from https://keras.io

10. J, A.K., Trueman, T.E., Cambria, E.: A convolutional stacked bidirectional LSTM with a multiplicative attention mechanism for aspect category and sentiment detection. Cognitive Computation. 13, 1423–1432 (2021).