Parallelizing Predicting Ratings using Customer Feedback

Vaidehi Pareshkumar Parikh **Shaival Sujalkumar Shah** **Avinash Deyyam**

1 Introduction

Online shopping has become increasingly popular in recent years. Global e-commerce sales are expected to reach 6.3 trillion in 2023. This represents a growth of 10.4% from 2022. The United States is expected to be the largest e-commerce market in 2023, with sales of 1.1 trillion dollars. This growth is being driven by a number of factors, including the convenience and affordability of online shopping, as well as the wide variety of products that are available online[7].

1.1 Problem Statement

One of the challenges of online shopping is that it can be difficult to know what products are worth buying. This is where customer feedback comes in. Customer feedback can provide valuable information about the quality of products, and it can help shoppers make informed decisions about what to buy.

One way to use customer feedback is to predict item ratings. This can be done by using a variety of machine learning techniques. However, one of the challenges of predicting item ratings is that it can be computationally expensive. This is because the number of possible combinations of products and ratings is very large.

In this project, we will use a parallelized approach to predict item ratings. This will allow us to reduce the computational cost of the prediction process. We believe that our approach has the potential to improve the accuracy of item rating predictions.

1.2 Objective

The objective of this project is to develop a parallelized approach to predict item ratings using customer feedback. This will be done by first performing sentiment analysis on the review text and then leveraging that to predict ratings for unseen reviews. The performance of the approach will be evaluated on the Multilingual Amazon Reviews Corpus, and the results will be compared to the results of other approaches.

1.3 Key Benefits

- NLP-based analysis of customer reviews enables data-driven decision-making by providing valuable insights into customer sentiment.
- Understanding customer feedback through NLP analysis of reviews, businesses can proactively address issues faced by customers.
- Improved product performance: Predicting customer ratings using reviews allows businesses to make informed decisions to improve their product or service.

2 Approach

2.1 Dataset

We have made use of the **The Multilingual Amazon Reviews Corpus**[13] dataset.

The corpus contains reviews in English, Japanese, German, French, Spanish, and Chinese, collected from 2015 to 2019. Each record contains the review headline, review body, title, star rating, an anonymized reviewer ID, an anonymized product ID, and the coarse-grained product category (e.g., 'books', 'appliances', etc.) The corpus is balanced across the 5 possible star ratings.

The total size of the dataset is something like this:

Total Reviews – **6,900,886** (7M approx.)

- Train Set: **100,000** Reviews

- Test Set: **10,000** reviews

2.2 Evaluation Metrics

Here, we have decided to go forward with the **Mean Absolute Error** or the MAE, along with the Classification Accuracy as well. We have preferred to consider MAE over Accuracy, because star ratings for each review are ordinal, and a 2-star prediction for a 5-star review should be penalized more heavily than a 4-star prediction for a 5-star review.

Classification accuracy ignores the ordinal nature of the labels.

Also, MAE is a simple and intuitive measure of error. It is easy to understand and interpret. It is also relatively robust to outliers.

Mean Absolute Error:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

y_i is the ground truth value for the i^{th} instance
y^i is the predicted value for the i^{th} instance
n is the number of instances

Next, we performed some Exploratory Data Analysis to get to know our dataset at hand and learn the nature of it.

2.3 Exploratory Data Analysis(EDA)

We will start off by getting to know our class labels.

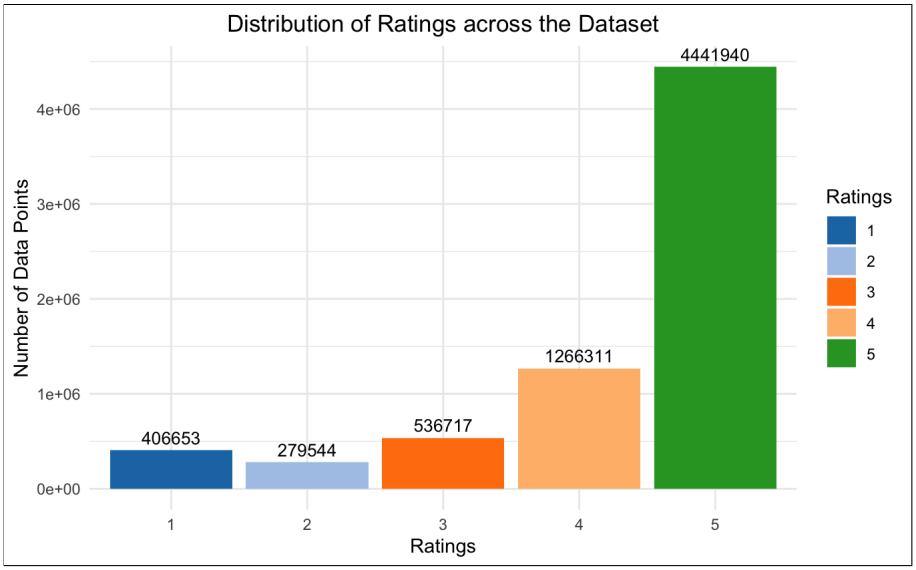

Figure 1: Rating distribution across dataset

As you can see from the image, the data is heavily imbalanced. 4 and 5-star rating reviews take up a huge proportion of the total reviews. This can be attributed to a classic phenomenon known as the purchasing bias. This just means that the people who are interested in a product, buy the product and have the opportunity to rate it. Customers who purchase a product are therefore more likely to positively review a product.

Instead of going with the classic re-sampling methods like class-weighting or SMOTE etc., we decided that we would be sampling our 100,000 rows in such a way that all 5-star ratings have an equal amount of reviews(i.e. 20,000 each). This way we would be completely eliminating the class imbalance.

Next, we will see here the division of reviews among various product categories.

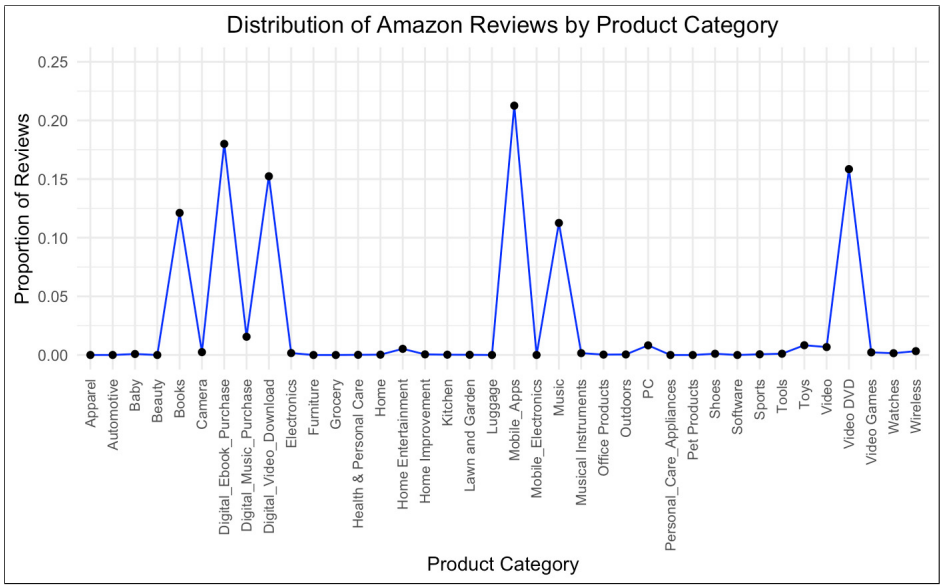

Figure 2: Review distribution across product categories

From the figure, we can deduce that hardly 5-6 product categories like Music, Mobile Apps, Mobile DVD that make up the majority chunk of the total reviews. This just shows that most of the people tend to buy these product categories only.

2.4 Data Preparation & Pre-Processing

We performed data preparation to ensure quality. We used **Spark SQL** to process the data and then shared it with our team. We all used the same pre-processed data for our modeling.

- Coming to the data preparation, we combined the **"review headline"** and the **"review body"**. This was done to consolidate relevant information into one field.This also helps to reduce missing out on relevant information present in the headline.
- We truncated the reviews to 4,000 characters to reduce the impact of long-range dependencies. This is important for models that rely on n-gram approaches, like word2vec or tf-idf, which can struggle to capture contextual meaning for longer texts. The shorter reviews are also easier to manage.
- We also utilized the vocab filter(i.e. Count Vectorizer) in order to filter out tokens that do not appear frequently in the dataset. Setting a minimum threshold like 20 times, helps remove infrequent words that may not carry meaningful information and might create noise into the analysis.

Moving on to the pre-processing part, attached here is the pre-processing pipeline that we followed in order to prepare it for the final stage of modeling.

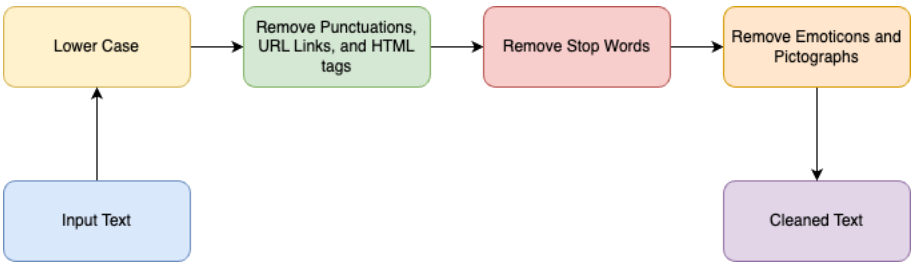

Figure 3: Pre-Processing Pipeline

The steps described above were used for all four models in the report.

We also performed **stemming** for the TF-IDF model. We **tokenized** the text for the Word2Vec model.

These were the two additional steps performed over the routine pipeline.

2.5 Modeling

2.5.1 Word2Vec + Random Forest Classifier:

Word2Vec[12] is an embedding technique that aims to represent words as continuous numerical vectors in a multi-dimensional space. It provides a continuous and dense representation for words, which can be used as features in text classification models.

It also captures semantic information and help in representing the contextual relationships between words.

A random forest classifier[8] is a type of ensemble learning model that is made up of a number of decision trees. Each decision tree is trained on a different subset of the training data, and the predictions of the individual trees are then combined to make a final prediction. The basic idea behind a random forest classifier is to use a large number of decision trees to make predictions.

Random forest classifiers are a powerful tool for multi-class text classification.

Some of the benefits for using RFC for such a task are:

- Robustness: Random forest classifiers are robust to noise and outliers in the training data.
- Interpretability: Random forest classifiers are relatively easy to interpret, which can be helpful for debugging and understanding the results of the model.
- Accuracy: Random forest classifiers can achieve high accuracy levels for multi-class text classification

2.5.2 TF-IDF + Multi-Layer Perceptron(MLP)

TF-IDF[11] is a popular feature representation technique used in NLP.

TF is a measure of how frequently a term appears in the document. It is the ratio of number of times a term appears in the document to total number of terms in the document.

IDF is a measure of how important a term is in the collection of documents. It is the ratio of total number of documents in the collection to the number of documents that contains the term at least once.

$$\mathbf{TF}(\mathbf{t}, \mathbf{d}) = \frac{\mathbf{freq}(\mathbf{t}, \mathbf{d})}{\mathbf{maxfreq}(\mathbf{d})}$$

where:
$\mathrm{freq}(t, d)$ is the frequency of the term t in the document d
$\mathrm{maxfreq}(d)$ is the frequency of the most frequent term in d.

$$\mathbf{IDF}(\mathbf{t}, \mathbf{D}) = \log \frac{\mathbf{N}}{\mathbf{n_t}}$$

where:
N is the total number of documents in D
n_t is the number of documents in D that contain the term t.

$$\mathbf{TFIDF}(\mathbf{t}, \mathbf{d}, \mathbf{D}) = \mathbf{TF}(\mathbf{t}, \mathbf{d}) \cdot \mathbf{IDF}(\mathbf{t}, \mathbf{D})$$

where:
t is a term (word) in the document
d is the document containing the term t
D is the collection of all documents

A Multi-Layer Perceptron (MLP)[9] is a type of artificial neural network that is commonly used for classification and regression tasks. MLPs are made up of a series of connected layers, each of which contains a number of neurons. The neurons in each layer are connected to the neurons in the next layer by weighted edges. The weights on the edges are learned during training, and they determine how the information flows through the network.

Here are some of the ways in which MLP is useful for a multi-class classification task:

- MLPs are able to learn nonlinear relationships between data. This is important for multi-class text classification, because the relationships between text and labels are often nonlinear.
- MLPs can be used with a variety of different features, and they can be adapted to a variety of different tasks.

2.5.3 Word-Piece Tokenizer (BERT) + Bi-directional LSTM

BERT tokenizer is used to preprocess text data before feeding it to the BERT model[14].

The BERT tokenizer uses WordPiece tokenization, which is a type of subword tokenization that breaks words into smaller pieces and represents each piece as a separate token. This allows the tokenizer to handle out-of-vocabulary words and improve the model's ability to generalize to new data.

For multi-class text classification, the BERT tokenizer can be used to tokenize the input text into a sequence of subword tokens, and then map these tokens to their corresponding indices in the BERT vocabulary. The resulting tokenized sequences can then be fed into the BERT model for classification.

Bi-directional LSTMs[2] are a type of recurrent neural network that is able to learn long-term dependencies in text data. This makes them well-suited for tasks such as text classification, where it is important to consider the context of the text.

Bi-directional LSTMs work by processing text in both directions, from left to right and from right to left. This allows them to capture information about both the past and the future of the text. This is important for text classification, because the meaning of a word or phrase can often depend on the context in which it is used. Benefits of using Bi-LSTMs are:

- Bi-LSTMs are able to model long-range dependencies in text data. This is because they can process text in both directions, from left to right and from right to left. This allows them to capture information about both the past and the future of the text.
- Bi-LSTMs are able to handle noisy data. This is because they are able to learn the underlying patterns in the data, even if the data is noisy.
- **Bi-LSTMs can be parallelized. This means that they can be trained on multiple GPUs at the same time. This can speed up the training process.**

2.5.4 DistilBERT (Base) + Neural Network

DistilBERT is a small, fast, and efficient Transformer model that is trained by distilling BERT base. It has 40% fewer parameters than BERT base, runs 60% faster, and achieves over 95% of BERT's performance[1].

To use this model, we first tokenized the input text using the "DistilBERT tokenizer", which splits the text into a sequence of subword tokens and maps each token to its corresponding index in the DistilBERT vocabulary. You can then feed the resulting tokenized sequence into the "DistilBertForSequenceClassification" model, which will generate a sequence of contextualized embeddings for each token in the sequence[4].

BERT is known for its state-of-the-art performance on a wide range of NLP tasks, including sequence classification. However, fine-tuning BERT can be computationally expensive, and it requires a large amount of labeled data to achieve optimal performance.

For sequence classification tasks, BERT first encodes the input sequence into a sequence of contextualized embeddings. These embeddings are then used as input to a classifier to predict the class label of the sequence.

3 Experiments and Results

3.1 Frameworks

- We used the **PySpark Framework** to train the Word2Vec + Random Forest and TF-IDF + Multi Layer Perceptron models. The PySpark framework is a distributed processing framework that is built on top of Apache Spark. It is designed to handle large datasets and to scale to multiple machines. In this project, we used PySpark to train the models on a cluster of machines with 32 GB of RAM and 8 cores each.
- We used the **PyTorch Framework** to train the DistilBERT + NN model. The PyTorch framework is a deep learning framework that is built on top of Python. It is designed to be easy to use and to scale to large datasets. In this project, we used PyTorch to train the model on a GPU node with 40 GB of RAM and an NVIDIA A100 GPU[5].
- We used the **TensorFlow Framework** to train the Bi-directional LSTM model. The TensorFlow framework is a deep learning framework that is built on top of Python. It is designed to be scalable and to be able to handle large datasets. In this project, we used TensorFlow to train the model on a GPU node with 64 GB of RAM and a p100 GPU.

3.2 Validation Analysis

In this project, we used K-fold cross-validation to tune the hyperparameters of our models to achieve optimal performance.

3.2.1 K-fold Cross Validation

K-fold cross-validation[10] is a statistical method used to evaluate machine learning models. It works by dividing the data into k folds, and then using each fold as a test set once. The model is trained on the remaining k-1 folds, and then its performance is evaluated on the test fold. This process is repeated k times, and the average performance of the model is reported.

K-fold cross-validation is a more robust way to evaluate a model than using a single test set. This is because it helps to reduce the variance in the model's performance. For example, if the model is trained on a single test set, and that test set happens to be biased, then the model's performance will be overestimated. However, if the model is evaluated on k folds, then the bias in the test set will be averaged out, and the model's true performance will be more accurately estimated.

Here, we made the choice of selecting **5 folds.**

3.2.2 Model Hyperparameters

The table below shows the optimal hyperparameters that we found for each model.

Hyperparameters	Value
vectorSize	100
numTrees	150
maxDepth	15
maxBins	64
impurity	gini

(a) Word2Vec + Random Forest Classifier

Hyperparameters	Value
layers	[200,100,50,5]
epochs	150
learning rate	0.02
batch_size	128
convergence tolerance	0.001

(b) TF-IDF + Multilayer Perceptron

Hyperparameters	Value
sequence length	128
n_layers	6
epochs	5
loss	sparse_categorical_crossentropy
optimizer	adam
LSTM units	64

(c) Word-Piece Tokenizer(BERT) + Bi-directional LSTM

Hyperparameters	Value
sequence length	512
n_layers	7
epochs	15
batch_size	64
loss	sparse_categorical_crossentropy
optimizer	adam

(d) DistilBERT + Neural Network

3.3 Validation Results

Table below shows the validation results for the selected hyperparameters:

Table 1: Validation Results

Model	Validation MAE	Validation Classification Accuracy
TF-IDF + Multi Layer Perceptron	0.98	42.5
Word2Vec + Random Forest Classifier	0.67	51.71
WordPiece Tokenizer(BERT) + Bi-Directional LSTM	0.373	74.58
DistilBERT-Base + Neural Network	0.21	84.1

3.4 Test Results

Table 2: Test Results

Model	Test MAE	Test Classification Accuracy
TF-IDF + Multi Layer Perceptron	0.99	42.3
Word2Vec + Random Forest Classifier	0.68	50.97
WordPiece Tokenizer(BERT) + Bi-Directional LSTM	0.379	73.01
DistilBERT-Base + Neural Network	**0.21**	**84.0**

As you can see from the table, DistilBERT achieved the lowest MAE and the highest classification accuracy.

3.5 Performance Analysis

Here, we will also be comparing the times taken with parallelism and without parallelism for each model. Firstly, the times for **TRAINING**:

Table 3: Training Times

Model	Time with Parallelism (mins)	Time w/o Parallelism (mins)
TF-IDF + Multi Layer Perceptron	1.28	3.12
Word2Vec + Random Forest Classifier	0.53	1.5
WordPiece Tokenizer(BERT) + Bi-Directional LSTM	3.5	10
DistilBERT-Base + Neural Network	9.5	2500

Now, we will compare times during **INFERENCE**:

Table 4: Inference Times

Model	Time with Parallelism (mins)	Time w/o Parallelism (mins)
TF-IDF + Multi Layer Perceptron	0.01	0.03
Word2Vec + Random Forest Classifier	0	0
WordPiece Tokenizer(BERT) + Bi-Directional LSTM	0.01	0.1
DistilBERT-Base + Neural Network	0.07	25

As you can see from the tables, parallelism significantly reduced the training time, but had little impact on the inference time.

PySpark

- Models that use parallelism can be trained in almost half the time of models that do not use Parallelism.
- Parallelism did not have a significant impact on inference time.
- Increasing the dataset size increases the efficiency of Parallelism.
- Increasing the number of cores decreased the training time.

PyTorch

- We trained our models on both CPU and GPU. The training speed with GPU was 230 times faster than with CPU[3].
- The training speed with GPU was far superior (230x) than that of CPU.

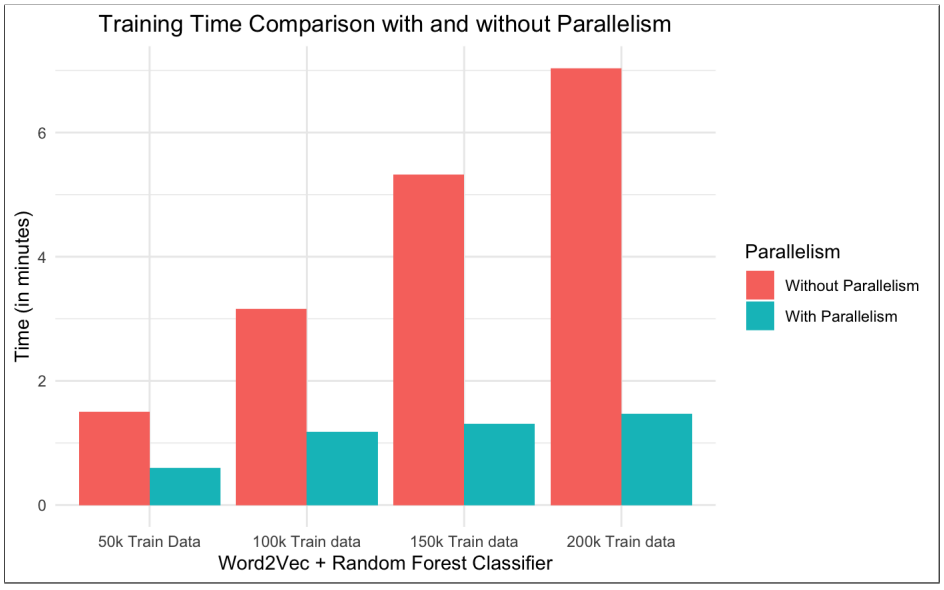

Figure 5: Training time comparison for RFC

The figures above shows that the training time with parallelism is significantly less than the training time without parallelism, and the gap between the two times widens as the dataset size increases. This is because parallelism is most effective when training with large datasets.

When training with a small dataset, the overhead of distributing the data across multiple cores can outweigh the benefits of parallelism. However, as the dataset size increases, the benefits of parallelism outweigh the overhead. This is because the time it takes to distribute the data across multiple cores becomes a smaller fraction of the total training time.

As a result, parallelism can significantly reduce the training time for large datasets. For example, in our experiments, we found that training a model with 100,000 examples took 8 times longer without parallelism than with parallelism.

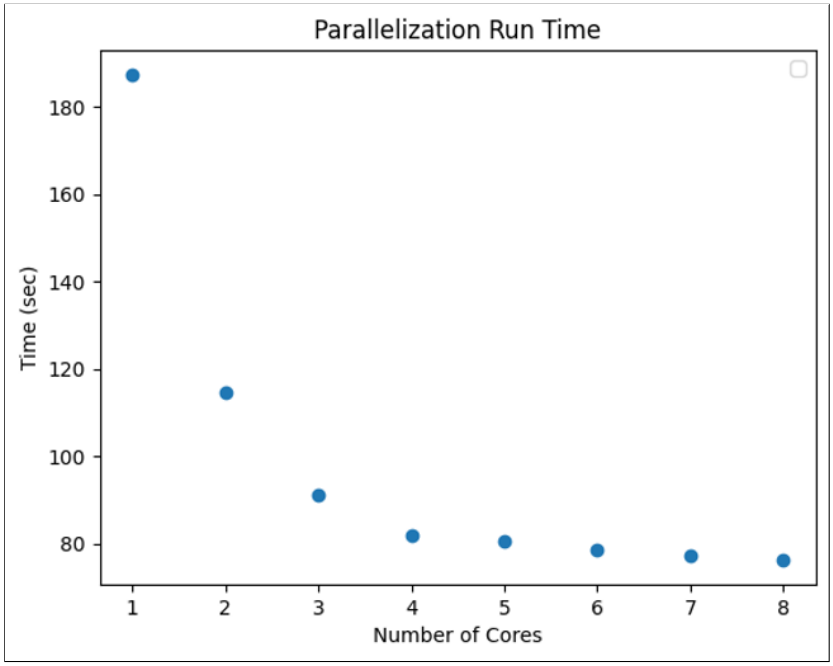

Figure 6: Training time with respect to number of cores

Also, while making use of the Spark framework we can see that as we increase the number of Partitions(cores) from 1 to 8, the time taken for training decreases drastically and becoming sort of constant after we increase the cores from 5 to 8.

4 Conclusion and Future Work

4.1 Conclusion

In this project, we have parallelized the prediction of star ratings using customer feedback. We have used the Amazon's customer reviews dataset in the US Marketplace and made use of PySpark, Pytorch and Tensorflow modules. We compared the performance of four different models: Random Forest Classifier, Multilayer Perceptron, Bidirectional LSTM, and DistilBERT. DistilBERT outperformed the other models with the highest classification accuracy and the lowest mean absolute error.

We also explored the benefits of parallelizing the prediction process using multiple CPU cores. This can significantly reduce the prediction time for large datasets. Overall, our project has demonstrated the effectiveness of using advanced machine learning models such as DistilBERT for multi-class text classification. We have also highlighted the importance of considering parallelization techniques for optimizing the prediction process. This project has practical applications in industries such as e-commerce and marketing, where understanding customer sentiment is crucial for business success.

4.2 Future Scope

- We plan to balance the dataset using re-sampling methods such as SMOTE. This will improve the performance of our models by reducing the bias in the dataset.
- We plan to build a model that includes additional features, such as helpful votes and total votes. We believe that this will improve the performance of our models by allowing them to better understand the sentiment of the reviews.
- We plan to utilize multi-GPU architecture for PyTorch and a multi-node setup for Spark. We believe that this will significantly reduce the training time for our models.

Here is some additional information about multi-GPU architecture and multi-node setup:

- **Multi-GPU architecture** allows you to use multiple GPUs to train your models. This can significantly reduce the training time, especially for large models.
- **Multi-node setup** allows you to train your models on multiple machines. This can also significantly reduce the training time, especially for very large models.

5 Statement of Contribution

1. Vaidehi Pareshkumar Parikh - Data Preprocessing, Word2Vec + Random Forest Classifier
2. Shaival Sujalkumar Shah - Data Preparation, Word-Piece Tokenizer(BERT) + Bi-directional LSTM
3. Avinash Deyyam - EDA, TF-IDF + Mutli-layer Perceptron, cross validation for DistilBERT (Base) + Neural Network
4. Bhanu Sai Simha Vanam - Data Preprocessing, DistilBERT (Base) + Neural Network

References

[1] https://arxiv.org/pdf/1910.01108.pdf

[2] https://arxiv.org/pdf/1801.01078.pdf

[3] https://cse.buffalo.edu/ demirbas/publications/DistMLplat.pdf

[4] https://huggingface.co/docs/transformers/tasks/sequence_classification

[5] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf

[6] /nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

[7] https://matheo.uliege.be/bitstream/2268.2/2707/4/Memoire_MarieMartin_s112740.pdf

[8] https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.classification.RandomForestClassifier.html

[9] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.MultilayerPerceptronClassifier.html

[10] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html?highlight=crossvalidator

[11] https://spark.apache.org/docs/latest/mllib-feature-extraction.html

[12] https://spark.apache.org/docs/3.1.2/api/python/reference/api/pyspark.ml.feature.Word2Vec.html

[13] https://arxiv.org/pdf/2010.02573.pdf

[14] https://arxiv.org/pdf/1810.04805.pdf