Mini Project 2 Report

Maša Ćirković

March 7, 2025

1 Introduction

Sentiment analysis plays an important role in understanding opinions, emotions, and feedback from textual data, which has wide applications in customer service, market research, and social media monitoring. By accurately classifying sentiment from text, businesses can gain insights into consumer behavior and public opinion, driving better decision-making.

This report focuses on constructing three distinct machine learning models for sentiment classification using the IMBD movie reviews dataset. The dataset contains textual reviews or feedback, which are analyzed to predict the sentiment behind them, and labels marking them as positive or negative reviws. Different machine learning approaches are explored, including traditional methods like Naïve Bayes and Linear Regression, as well as advanced deep learning technique with Convolutional Neural Network (CNN) model.

The goal is to evaluate and compare the performance of each model, identifying the most effective approach for accurately classifying sentiment. This analysis aims to demonstrate how various machine learning methods can be applied to sentiment classification.

2 Data Processing

Data Processing involves understanding what the data represents, exploring different data properties and transforming it into a suitable representation for further analysis or training the models.

2.1 Data Explanation

The dataset can be explored at this link. It contains reviews about movies and a label which represents whether the review is positive or negative. Negative review is represented with 0, while positive is represented with 1. Dataset structure is given in 1.

Dataset Part	**Number of examples**
Train	25,000
Test	25,000
Unsupervised	50,000

Table 1: *Description of the IMDB dataset*

2.2 Approach

The first part was standardizing the reviews, which included removing html tags, punctuations, numbers, single characters and multiple spaces to allow models to focus on the content rather than structure. This was demonstrated during the hands-on practice with sentiment analysis.

Next, null values for the dataset were checked and there were none.

In order to get insight into the distribution of reviews, their labels were plotted and shown in the figure 1. As it can be seen, this dataset is perfectly balanced with 50% of the data belonging to the negative class, and 50% of the data belonging to the positive class.

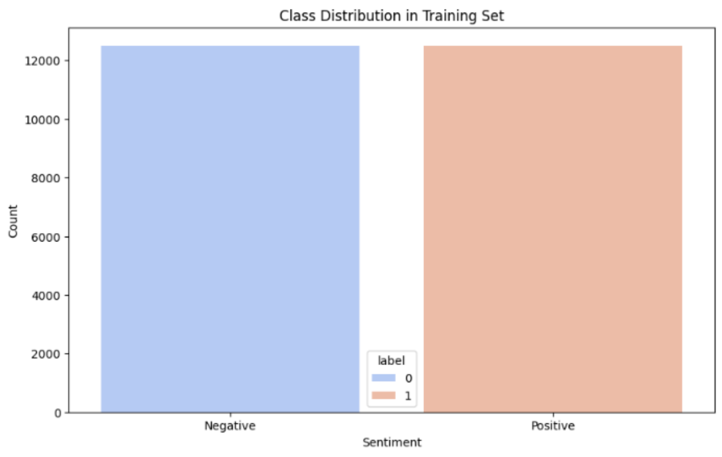


Figure 1: *Class distribution of positive versus negative reviews*

Lastly, review length distribution was explored for both training and testing data and added as a column to the dataframe. It can be seen from figure 2 that the majority of the reviews have length of around 200 words, and that the distribution is right-skewed.

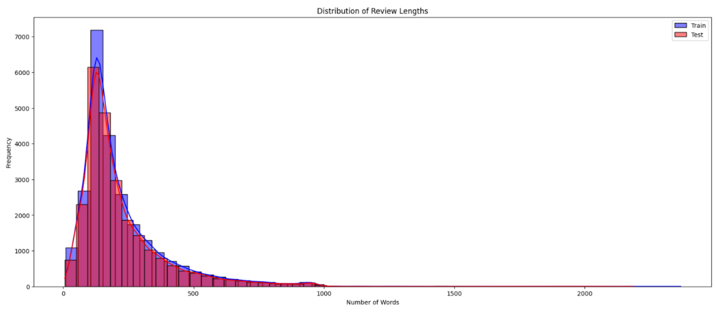


Figure 2: *Distribution of reviews lengths of train and test datasets*

3 Modeling

Three models were chosen for this project: Linear Regressor, Naive Bayes, and Convolutional Neural Network (CNN). In order to be able to recreate results, random seed was set to 42 for two machine learning models and for tensorflow, however it is still not guaranteed that CNN will produce the same results.

Since the loaded data was already split into train and test, no additional work on that part has been done.

Models were compared based on accuracy they achieved for this classification task.

Accuracy (Eq. 1) is a fundamental metric used to evaluate the performance of classification models. It measures the proportion of correctly predicted examples out of all examples in the dataset.

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} \mathbf{1}(y_i = \hat{y}_i)}{n} \tag{1}$$

- y_i is the actual class label.
- $\hat{y}_i$ is the predicted class label.
- n is the total number of samples.
- $\mathbf{1}(y_i = \hat{y}_i)$ is an indicator function that equals 1 if the prediction is correct and 0 otherwise.

Accuracy provides a simple and intuitive measure of a model's performance, where a higher value indicates better classification performance. However, it may not be suitable for imbalanced datasets, where other metrics like precision, recall, and F1-score might be more informative. Since this dataset is perfectly balanced, accuracy is a good measure.

In order to train two simple machine learning models, Linear Regressor and Naive Bayes, Term Frequency - Inverse Document Frequency (TF-IDF) was used.

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (in this case importance of a word in a review relative to the collection of reviews). It consists of two components:

1. **Term Frequency (TF)**: The term frequency measures how frequently a word appears in a document:
$$TF(w) = \frac{\text{Num. of times word } w \text{ appears in document}}{\text{Total num. of words in the document}}$$
2. **Inverse Document Frequency (IDF)**: The inverse document frequency measures how unique or rare a word is across all documents:
$$IDF(w) = \log \frac{\text{Total num. of documents}}{\text{Num. of documents containing word } w}$$
- Common words (e.g., "the", "is") have lower IDF because they appear in many documents. - Rare words have higher IDF, giving them more importance.
3. **Final TF-IDF Score**: The final TF-IDF score is given by:
$$TF\text{-}IDF(w) = TF(w) \times IDF(w)$$
This helps assign higher importance to key words while reducing the influence of frequent, unimportant words.

TF-IDF is commonly used for traditional machine learning models such as Naïve Bayes and Logistic Regression because these models expect numerical inputs rather than raw text, which is the way reviews are.

Naïve Bayes assumes that word occurrences are independent given the class. TF-IDF helps by:

- Reducing the impact of common words.
- Assigning higher importance to rare but significant words.
- Improving the model's ability to distinguish between different classes.

Logistic Regression is a linear classifier, meaning it finds a decision boundary based on weighted input features. TF-IDF is useful because:

- It normalizes the text data.
- It gives more weight to important words while reducing noise from common words.
- It improves classification performance compared to raw word counts.

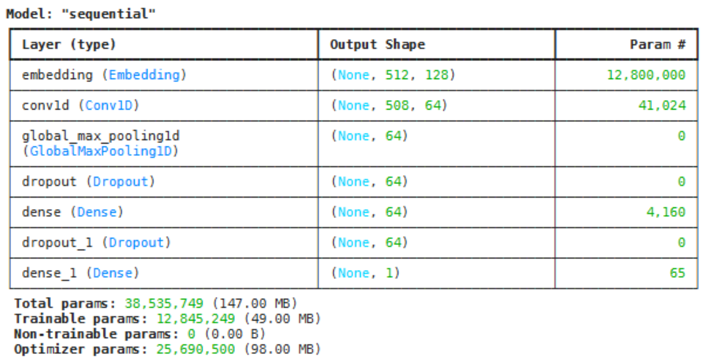
CNN model requires some tokenization and padding to be done beforehand. Specifically, maximum vocabulary size (the number of unique words to consider) and maximum length of text sequences need to be set. In this project for maximum vocabulary size 100,000 was used, and 512 for maximum length. These numbers were chosen as a trade-off based on these characteristic of the dataset:

- *Maximum review length*: 2368
- *Average review length*: 220.35
- *Percentage of reviews with $\leq$ 512*: 93.21
- *Length of 95th percentile*: 573
- *Total unique words*: 264870

As these two parameters increase, so does the complexity and the time it takes. Strategy for dealing with longer reviews is to truncate them at the end, and for shorter reviews padding is used.

The CNN architecture is presented in the figure 3. Convolutional layer uses a L2 regularizer with 0.01. Hyperparameters used are:

- *Embedding dimension*: 128
- *Number of filters*: 64
- *Size of each filter*: 5
- *Dropout rate*: 0.7



```
Model: "sequential"
┌──────────────────────────────────┬──────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape             │      Param # │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ embedding (Embedding)            │ (None, 512, 128)         │   12,800,000 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ conv1d (Conv1D)                  │ (None, 508, 64)          │       41,024 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ global_max_pooling1d             │ (None, 64)               │            0 │
│ (GlobalMaxPooling1D)             │                          │              │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dropout (Dropout)                │ (None, 64)               │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 64)               │        4,160 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dropout_1 (Dropout)              │ (None, 64)               │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense_1 (Dense)                  │ (None, 1)                │           65 │
└──────────────────────────────────┴──────────────────────────┴──────────────┘
 Total params: 38,535,749 (147.00 MB)
 Trainable params: 12,845,249 (49.00 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 25,690,500 (98.00 MB)
```

Figure 3: *Structure of the CNN model*

3.1 Results

Since traditional machine learning methods don't have training epochs, their learning curve is represented using increasing size of training data. Specifically, the dataset size started at 60% and had 10 steps up to 100%. Cross-validation is used to obtain the learning curve and StratifiedKFold was used to ensure class balance when creating subsets of dataset. Learning curve visualizes training and validation accuracy on increasing dataset size.

Learning curve for Logistic Regression is given in the figure 4. As it can be seen, the training accuracy is very steady across increasing dataset sizes, meaning that this model is robust in regards to training size. Validation accuracy is steadily increasing as the number of samples increases.

Logistic Regression achieved accuracy of **88.256%**.

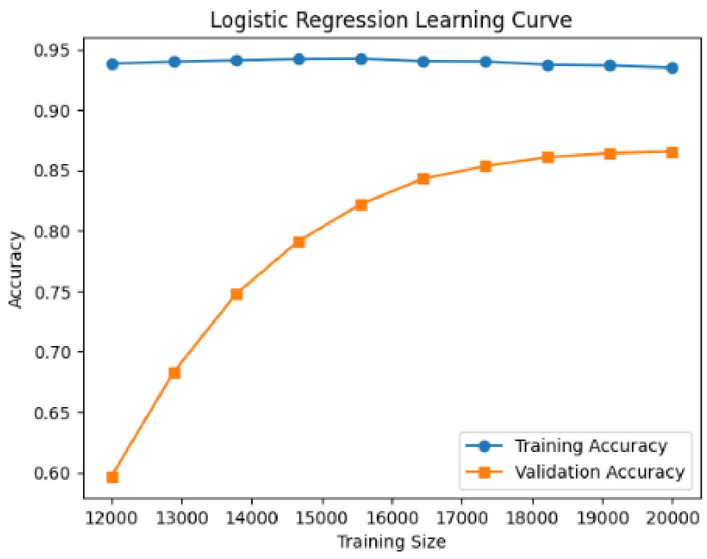


Figure 4: *Logistic Regression learning curve*

Learning curve for Naive Bayes is given in the figure 5. This curve is more interesting to discuss. It appears that there is a certain range of training sizes where the algorithm doesn't perform well for training accuracy, but after passing that point it starts to steadily increase. Validation accuracy is very low up until that point as well, then starts to increase together with training accuracy.

Naive Bayes achieved accuracy of **83.104%**

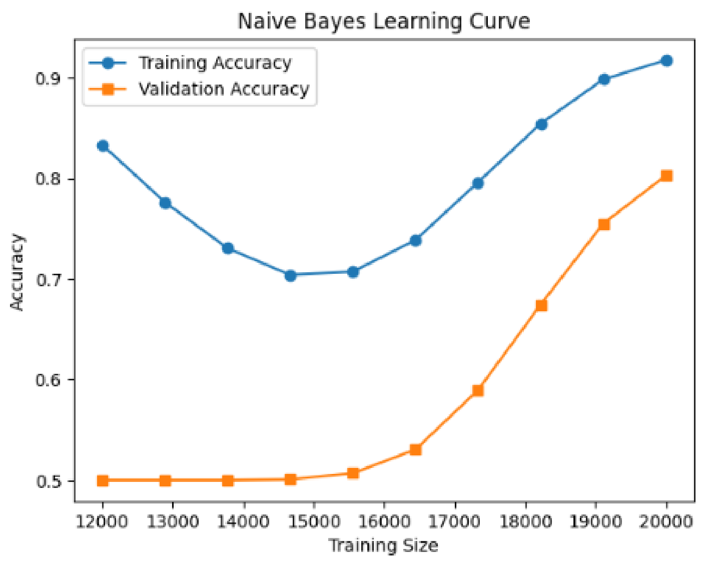


Figure 5: *Naive Bayes learning curve*

CNN model was trained with early stopping based on validation accuracy. It was trained for 10 epochs with batch size of 64. Accuracy learning curve is given in the figure 6, while loss learning curve is given in the figure 7. These curves suggest overfitting, since training loss keeps decreasing and training accuracy keeps increasing, but validation loss decreases until one point and then stars increasing, and validation accuracy keeps increasing until one point but then starts decreasing. Model parameters can be changed, or more data can be used for training in order to deal with this problem.

With all of this, CNN model achieves validation accuracy of **84.94%**.

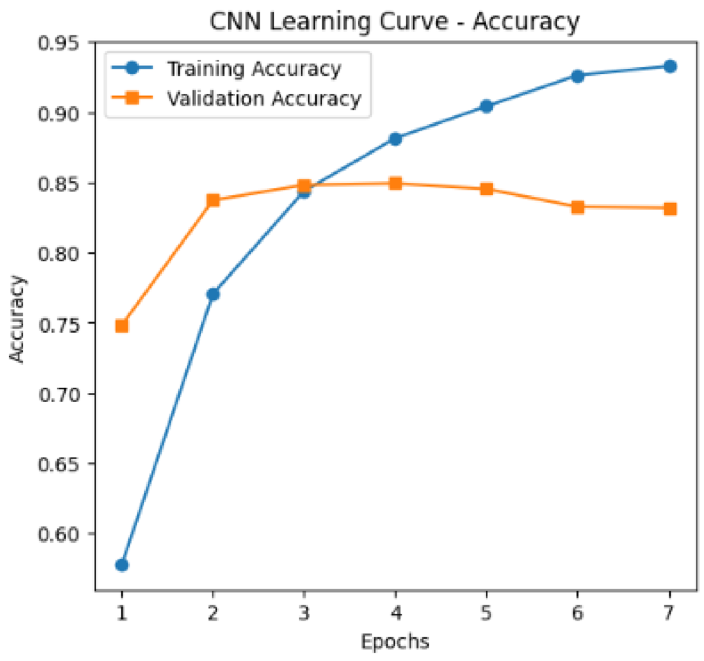


Figure 6: *CNN model accuracy learning curve*

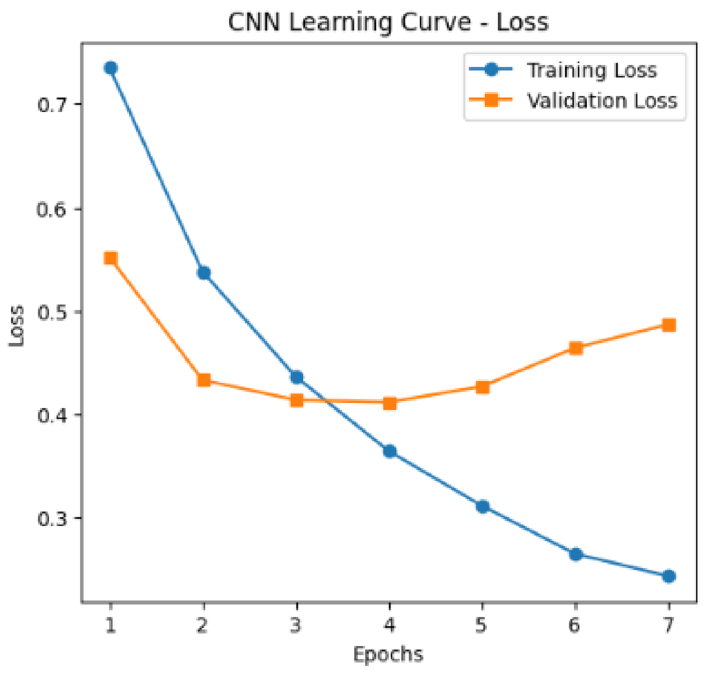


Figure 7: *CNN model loss learning curve*

All of the models achieve over 80% accuracy. The difference in their results comes from the underlying architecture. In general, Deep Learning (DL) models should outperform traditional models, but training them can be difficult because of hyperparameter tuning and the amount of data they need to work properly.

3.2 Visualization

Confusion matrix and Receiver Operating Characteristic (ROC) curve for each model will be shown here.

Confusion matrices are shown in figures 8, 9, 10. As it can be seen, the Logistic Regression model makes the most balanced predictions, both in terms of correctly predicted and not correctly predicted classes. Naive Bayes better predicts true negative class compared to true positive, and makes more mistakes regarding false negative than false positive, suggesting that the positive reviews are not that easy to understand. CNN model also makes balanced true positive and true negative predictions, but makes more mistakes when it comes to false negative than false positive. This can be due to truncation or not enough examples or not the best model parameters.

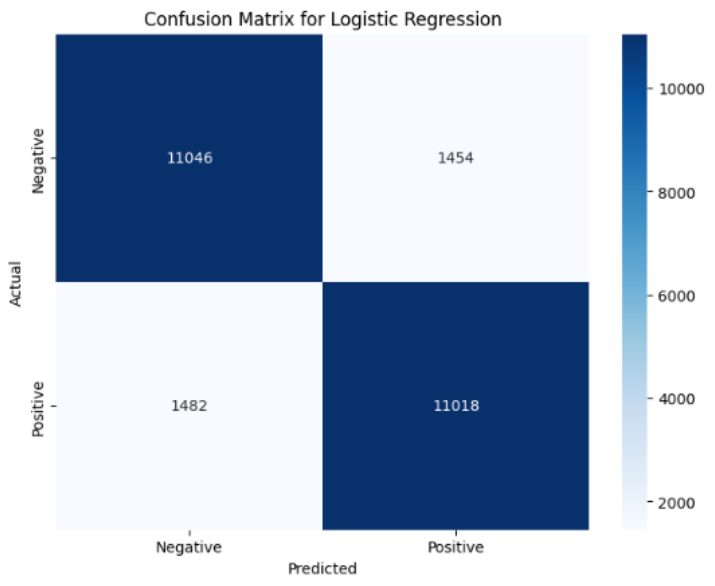


Figure 8: *Confusion matrix for Linear Regression*

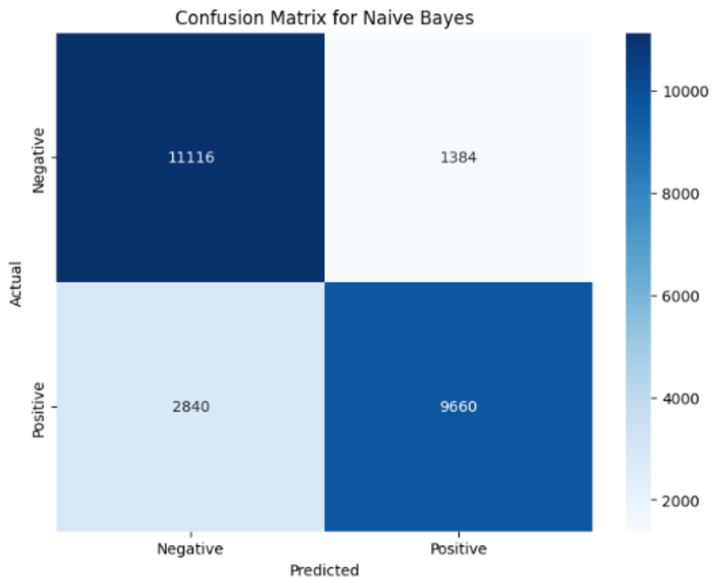


Figure 9: *Confusion matrix for Naive Bayes*

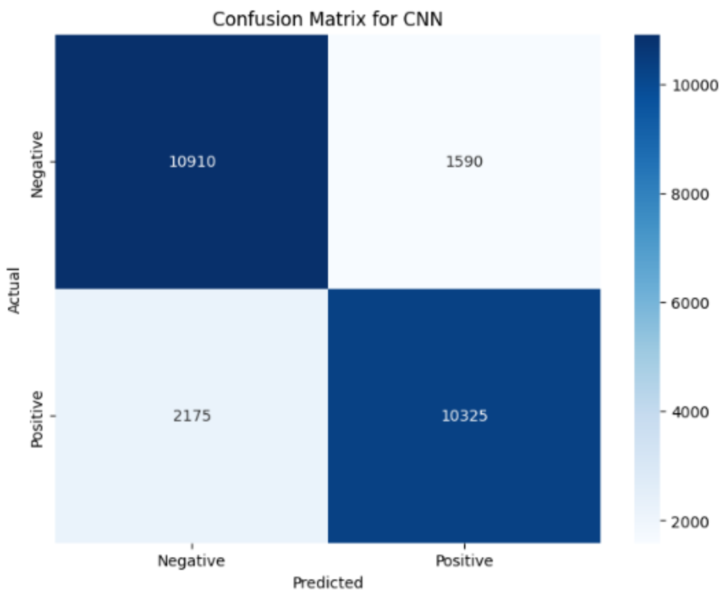


Figure 10: *Confusion matrix for CNN model*

ROC curves are shown in figures 11, 12, 13. All of the models have a high Area Under Curve (AUC) score, suggesting that the models are very effective and have a good discrimination.

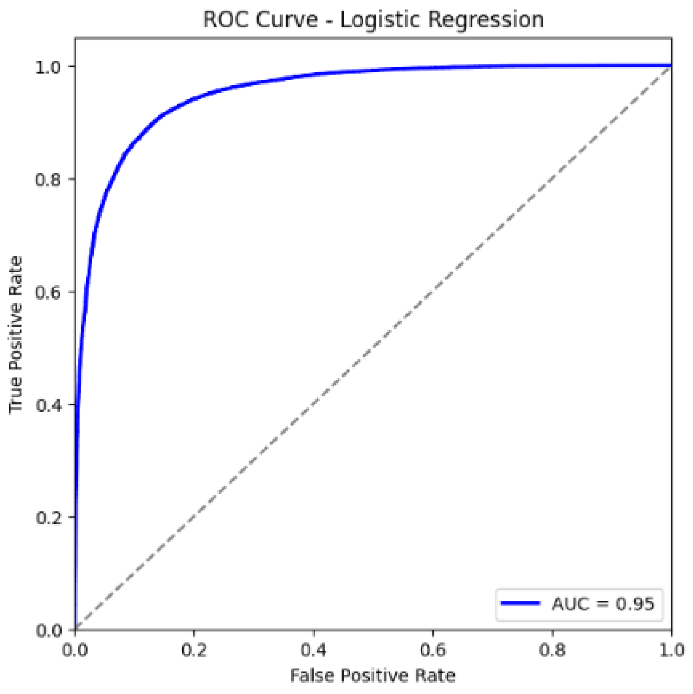


Figure 11: *ROC curve for Linear Regression*

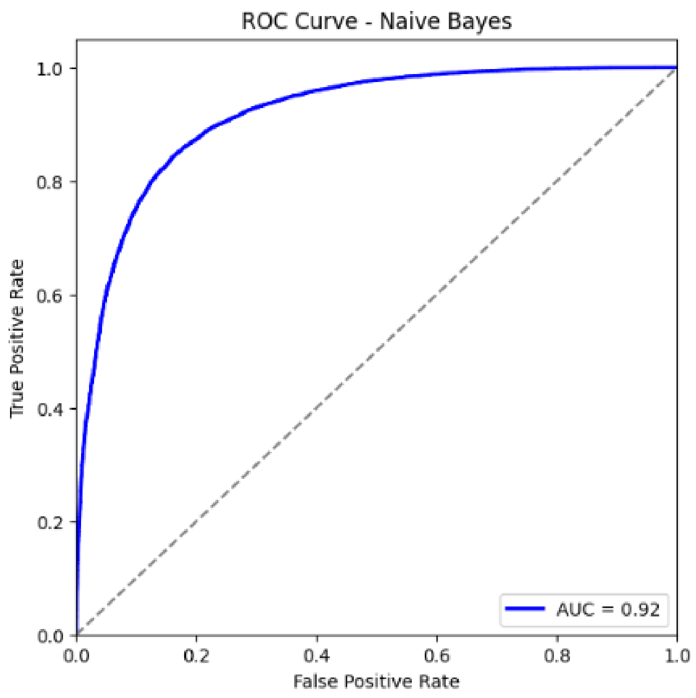


Figure 12: *ROC curve for Naive Bayes*

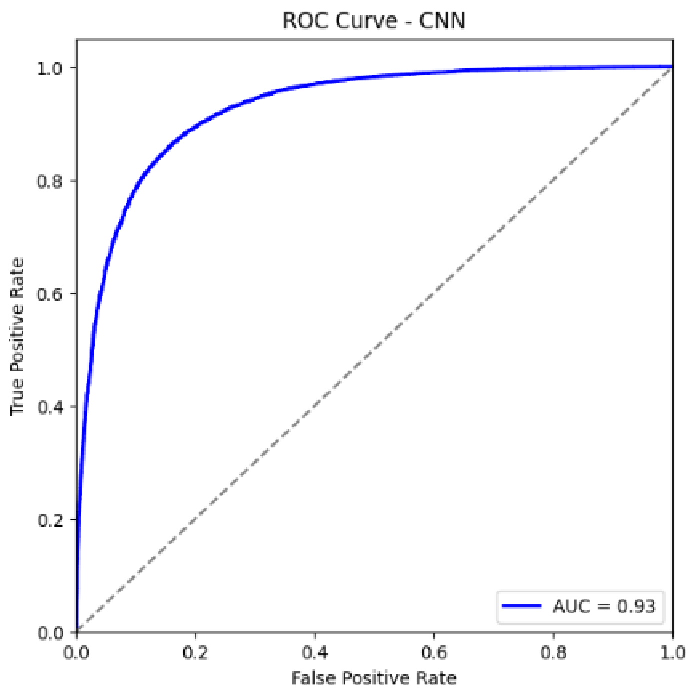


Figure 13: *ROC curve for CNN model*

4 Conclusion

Best performing model in this case was Linear Regression, with accuracy score of 88.256%, and it also has the most balanced classification both in terms of correctly predicted classes, and in terms of incorrectly predicted classes.

The biggest challenge here was training the DL model, since it requires a lot of trial and error with hyperparameters, but also since it requires a lot of data. With a bigger amount of data and more testing with hyperparameters, it would outperform other models easily, but that was not the point of this project. Some other reasons why it might be performing better are given below:

- **Linearity in the data**: If the relationship between features and the target variable is approximately linear, Linear Regression will naturally perform well. Naïve Bayes assumes conditional independence of features, which might not hold, leading to poor generalization. CNNs are better suited for spatial data and may not perform well on textual data.
- **CNN struggles with small datasets**: CNNs require large datasets to learn hierarchical patterns. If the dataset is small, CNNs may overfit (as it was in this case), whereas simpler models like Linear Regression generalize better..

In conclusion, sometimes a simpler model might outperform more complex models, so we should not use a DL model for every problem presented to us, rather we should think about whether it is needed at all and what are the trade-offs we get.