CS771: CHITTI

Lakshyta Mahajan 220581

Umang Garg 221156

Navnit Patel 220701

Contents

1 Introduction

In this mini-project, we explore the task of **binary classification** using three distinct datasets derived from the same underlying dataset. Each dataset represents different feature representations of the same input data, posing a challenge in selecting suitable machine learning (ML) models. The goal of this project is to evaluate the performance of different ML models for each dataset and to determine if combining these datasets can improve performance beyond training models on individual datasets.

1.1 Motivation

Binary classification problems are prevalent in machine learning, and selecting the best feature representation is critical for improving model performance. Each dataset provided in this project corresponds to different representations—ranging from emoticon-based categorical features to embeddings generated from a deep neural network and sequence-based representations of input. The diversity in feature types challenges us to choose models that balance accuracy and training efficiency while minimizing computational complexity.

1.2 Project Overview

This mini-project focuses on training binary classifiers on three distinct datasets, with the goal of identifying the best-performing machine learning models based on their accuracy and efficiency in training. The datasets are:

- **Emoticon-based Features Dataset**: Each input is represented using 13 categorical emoticons.
- **Deep Features Dataset**: Each input consists of a 13x786 matrix generated using a deep neural network.
- **Text Sequence Dataset**: Each input is a sequence of 50 digits, providing a string-based representation.

Each dataset is split into training, validation, and test sets. The validation set is used to select hyperparameters and assess model performance, while the test set's ground-truth labels are hidden for external evaluation.

The primary objective is to identify the best-performing model for each dataset based on two criteria:

1. High validation and test accuracy.
2. Minimal training data utilization to reduce computational cost and increase generalization.

The project is divided into two main tasks:

- **Task 1**: Train binary classifiers on each of the three datasets independently and identify the best-performing models. The "best" model is defined as one that achieves a high level of accuracy on the validation and test sets, while also requiring a minimal amount of training data. We analyze the impact of training size by using various percentages of the data (20%, 40%, 60%, 80%, 100%) and assess the corresponding validation accuracies.
- **Task 2**: Combine the feature representations from the three datasets to build a single model, aiming to improve the accuracy over the individual dataset models. The combination of datasets poses an additional layer of complexity as it allows us to explore whether multi-representation data can lead to better generalization and higher predictive accuracy.

2 Task I

2.1 Part I

2.1.1 Data Preprocessing

Data preprocessing is a crucial step in preparing the emoticon dataset for analysis and model training. The preprocessing involved several key steps:

1. **Data Loading**: The emoticon datasets were loaded using `pandas` from CSV files. The datasets included training, validation, and test sets. Each dataset contained emoticons as features and their corresponding labels.
2. **Emoticon to Unicode Conversion**: Since emoticons are represented as characters, they were converted into their Unicode integer values. This conversion was performed using a custom function, `emoticons_to_unicode`, which iterated over each character in the emoticon string and obtained its Unicode representation using the `ord()` function.

3. **One-Hot Encoding**: To facilitate the machine learning model's understanding of the emoticons, One-Hot Encoding was applied. This process transformed the Unicode values into a binary matrix, where each column represented a unique emoticon and each row corresponded to an emoticon string. The `OneHotEncoder` from `scikit-learn` was utilized to achieve this.

4. **Train-Validation Split**: The training data was split into subsets for training and validation purposes. The validation set allowed for model evaluation during training, ensuring that overfitting could be monitored.

5. **Final Preparations**: The training and validation datasets were transformed into lists of encoded emoticons, ready to be fed into the machine learning model.

2.1.2 Training Methodology

The model implemented for this emoticon classification task was the **Multinomial Naive Bayes** classifier. This model was chosen due to its effectiveness in handling categorical data and its suitability for text classification tasks.

1. **Model Initialization**: The Multinomial Naive Bayes classifier was initialized with different values of the smoothing parameter, alpha. The alpha parameter helps prevent overfitting by smoothing the probabilities of features that may not appear in the training set.

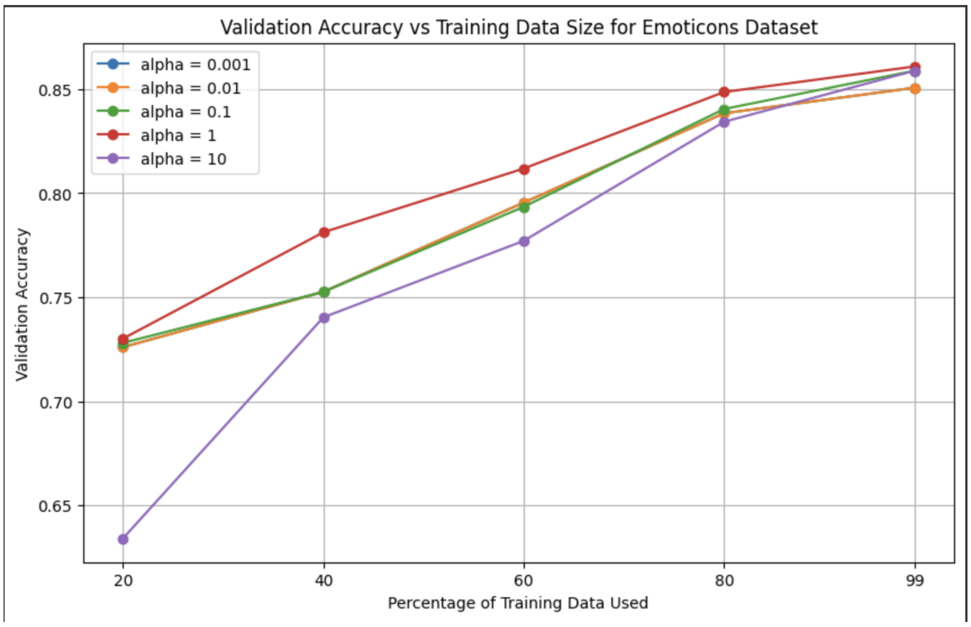
2. **Model Training**: The training methodology involved experimenting with various training sizes and alpha values. The training data was split into different sizes (20%, 40%, 60%, 80%, and 99% of the dataset). For each training size, the model was trained multiple times with different alpha values (0.001, 0.01, 0.1, 1, 10).

3. **Validation**: After training, the model was evaluated on the validation set to calculate accuracy. The performance metrics were recorded, allowing for comparison between different training sizes and alpha values. Notably, the model demonstrated the best validation accuracy when trained with 80% of the training data.

4. **Graphical Representation**: The accuracies for each combination of training size and alpha value were plotted, providing a visual representation of the model's performance across different configurations.

2.1.3 Model Evaluation

The evaluation of the Multinomial Naive Bayes model involved analyzing the accuracy metrics obtained from the validation set. The graph displaying accuracy vs. training data size for various alpha values will be inserted here:



This graph illustrates how the model's accuracy varied with different percentages of training data and alpha values, providing insights into the model's robustness and generalization capabilities.

2.1.4 Best Model Selection

After evaluating the model's performance across various configurations, the best model was selected based on the validation accuracy.

- The final chosen model utilized **Multinomial Naive Bayes** with **alpha = 1**.
- This configuration yielded the best accuracy on the validation set when trained with **80% of the training data** which does not increase much even by using 100% training data.

The selected model was then applied to the test dataset, and the predictions were saved for further analysis.

2.2 Part II

2.2.1 Data Preprocessing

Data preprocessing is a crucial step in preparing the emoticon dataset for analysis and model training. The preprocessing involved several key steps:

1. **Data Loading**: The emoticon datasets were loaded using `pandas` from CSV files. The datasets included training, validation, and test sets. Each dataset contained emoticons as features and their corresponding labels.
2. **Emoticon to Unicode Conversion**: Since emoticons are represented as characters, they were converted into their Unicode integer values. This conversion was performed using a custom function, `emoticons_to_unicode`, which iterated over each character in the emoticon string and obtained its Unicode representation using the `ord()` function.
3. **Input Transformation**: In this case, the input was transformed into a 13x376 matrix, where each row represented a sequence of Unicode values corresponding to an emoticon. This structure allowed for more complex relationships to be captured during model training.
4. **One-Hot Encoding**: To facilitate the machine learning model's understanding of the emoticons, One-Hot Encoding was applied. This process transformed the Unicode values into a binary matrix, where each column represented a unique emoticon and each row corresponded to an emoticon string. The `OneHotEncoder` from `scikit-learn` was utilized to achieve this.
5. **Train-Validation Split**: The training data was split into subsets for training and validation purposes. The validation set allowed for model evaluation during training, ensuring that overfitting could be monitored.
6. **Final Preparations**: The training and validation datasets were transformed into lists of encoded emoticons, ready to be fed into the machine learning model.

2.2.2 Training Methodology

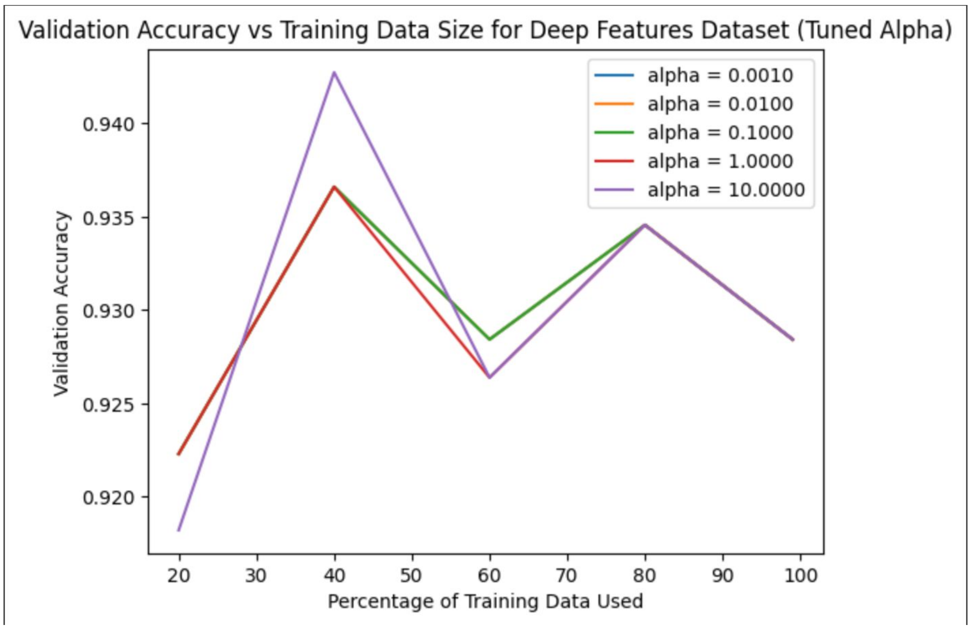
The model implemented for this emoticon classification task was the **Multinomial Naive Bayes** classifier. This model was chosen due to its effectiveness in handling categorical data and its suitability for text classification tasks.

1. **Model Initialization**: The Multinomial Naive Bayes classifier was initialized with different values of the smoothing parameter, alpha. The alpha parameter helps prevent overfitting by smoothing the probabilities of features that may not appear in the training set. In this case, the best hyper-parameter was found to be **alpha = 10**.
2. **Model Training**: The training methodology involved experimenting with various training sizes and alpha values. The training data was split into different sizes (20%, 40%, 60%, 80%, and 99% of the dataset). For each training size, the model was trained multiple times with different alpha values.

3. **Validation**: After training, the model was evaluated on the validation set to calculate accuracy. The performance metrics were recorded, allowing for comparison between different training sizes and alpha values. Notably, the model demonstrated the best validation accuracy when trained with **40%** of the training data.
4. **Graphical Representation**: The accuracies for each combination of training size and alpha value were plotted, providing a visual representation of the model's performance across different configurations.

2.2.3 Model Evaluation

The evaluation of the Multinomial Naive Bayes model involved analyzing the accuracy metrics obtained from the validation set. The graph displaying accuracy vs. training data size for various alpha values will be inserted here:



This graph illustrates how the model's accuracy varied with different percentages of training data and alpha values, providing insights into the model's robustness and generalization capabilities.

2.2.4 Best Model Selection

After evaluating the model's performance across various configurations, the best model was selected based on the validation accuracy.

- The final chosen model utilized **Multinomial Naive Bayes** with **alpha = 10**.
- This configuration yielded the highest accuracy on the validation set, particularly when trained with **40%** of the training data.

The selected model was then applied to the test dataset, and the predictions were saved for further analysis. The validation accuracy with the chosen parameters confirmed the model's effectiveness in classifying emoticons, establishing a reliable approach for future applications in similar tasks.

2.3 Part III

2.3.1 Data Prerprocessing

Data preprocessing is a crucial step in preparing the text sequence dataset for analysis and model training. The preprocessing involved several key steps:

1. **Data Loading**: The text sequence datasets were loaded using `pandas` from CSV files. The datasets included training and validation sets, each containing input strings and their corresponding labels.

2. **Input Transformation**: The input strings were transformed into sequences of integers (digit encoding) to represent each character. This was achieved through a custom preprocessing function, converting each character in the input strings into its integer representation.

3. **Padding Sequences**: To ensure uniform length across sequences, padding was applied. Each input string was padded to a maximum length of 50, allowing the model to handle variable-length sequences effectively.

4. **Label Encoding**: The labels were converted into categorical format for multi-class classification, enabling the model to interpret the output labels correctly.

5. **Final Preparations**: The training and validation datasets were transformed into numpy arrays, ready to be fed into the RNN model.

2.3.2 Training Methodology

The model implemented for this text sequence classification task was a **Recurrent Neural Network (RNN)**. This model was chosen for its ability to capture temporal dependencies in sequence data.

1. **Model Initialization**: The RNN model was initialized with an embedding layer, followed by two SimpleRNN layers. The embedding layer transformed the integer-encoded input into dense vectors suitable for the RNN.

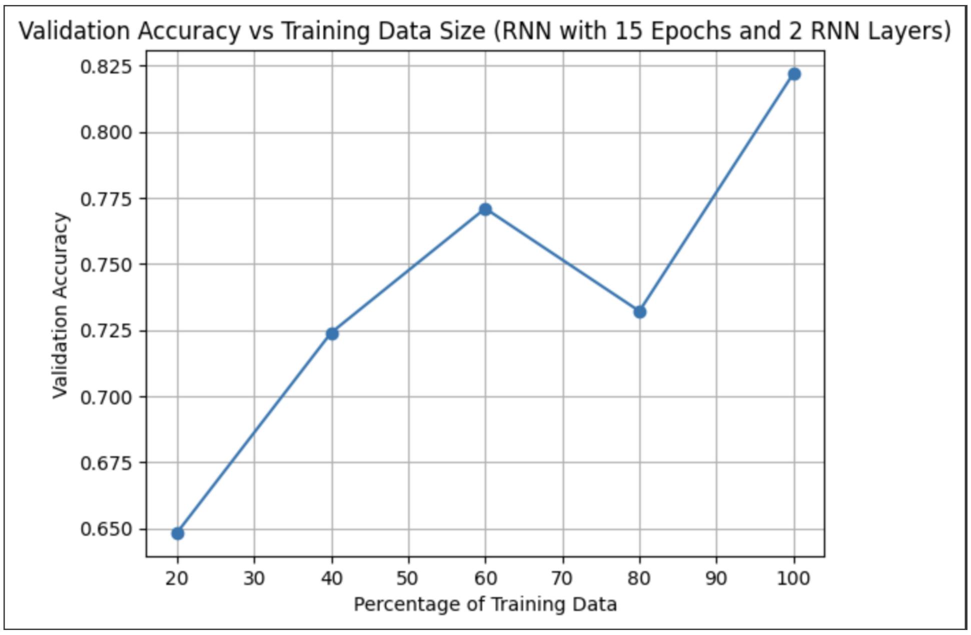
2. **Model Training**: The training methodology involved experimenting with different sizes of training data (20%, 40%, 60%, 80%, and 100%) over 15 epochs. The model was trained using the categorical cross-entropy loss function and the Adam optimizer.

3. **Validation**: After training, the model was evaluated on the validation set to calculate accuracy. The performance metrics were recorded, allowing for comparison between different training sizes. Notably, the model achieved the best validation accuracy with **100%** of the training data.

4. **Graphical Representation**: The accuracies for each combination of training size were plotted, providing a visual representation of the model's performance across different configurations.

2.3.3 Model Evaluation

The evaluation of the RNN model involved analyzing the accuracy metrics obtained from the validation set. The graph displaying accuracy vs. training data size for various configurations will be inserted here:



This graph illustrates how the model's accuracy varied with different percentages of training data, providing insights into the model's robustness and generalization capabilities.

2.3.4 Best Model Selection

After evaluating the model's performance across various configurations, the best model was selected based on the validation accuracy.

- The final chosen model utilized **Recurrent Neural Network (RNN)** architecture.
- This configuration yielded the highest accuracy on the validation set, particularly when trained with **100%** of the training data.

The selected model was then applied to the test dataset, and the predictions were saved for further analysis. The validation accuracy with the chosen parameters confirmed the model's effectiveness in classifying text sequences, establishing a reliable approach for future applications in similar tasks.

3 Task II

3.1 Data Preprocessing

The preprocessing steps for Task 2 involved similar procedures as in Task 1. We utilized three datasets: **emoticons**, **text sequences**, and **features**. Each dataset was prepared for model input, ensuring compatibility across the different data types. Emoticon and text sequence data were converted to numerical representations to facilitate processing by the machine learning algorithms.

3.2 Models and Training Methodology

For Task 2, we implemented two models to predict binary outputs (0 or 1) based on the combined input from all three datasets.

3.2.1 Neural Network Model

The model implemented for this task was a **neural network**, designed with the following structure:

1. **Model Structure**: The network consisted of multiple dense layers: one with 128 units, followed by 64 units, and then 32 units. The final output layer utilized the **softmax** activation function to handle binary classification.
2. **Model Training**: We trained the model using various percentages of the training data. The model yielded high accuracy but was computationally intensive, having over 10,000 trainable parameters. Notably, the accuracy of the validation set peaked when trained with **100%** of the training data, achieving a maximum validation accuracy of approximately **99%**.
3. **Validation**: After training, the model was evaluated on the validation set to calculate accuracy. The performance metrics were recorded, enabling comparison of accuracy across different training sizes.
4. **Graphical Representation**: The validation accuracies for each percentage of training data used were plotted, providing a visual representation of the model's performance.

3.2.2 Multinomial Naive Bayes Model

The second model employed was a **Multinomial Naive Bayes classifier**.

1. **Model Initialization**: The Multinomial Naive Bayes classifier was initialized with different values for the hyperparameter **alpha**, ranging from 0.1 to 2.0. This hyperparameter helps prevent overfitting by smoothing the probabilities of features that may not appear in the training set. The optimal performance was observed with **alpha = 0.1**.
2. **Model Training**: The training methodology involved experimenting with various training sizes and corresponding alpha values. For each training size, the model was trained multiple times with different alpha configurations.

3. **Validation**: After training, the model was evaluated on the validation set to compute accuracy. The performance metrics were recorded, allowing for comparison between different training sizes and alpha values. The model exhibited optimal performance when trained with **40%** of the training data yeilding an accuracy of approximately **86%**.
4. **Graphical Representation**: The accuracies for each combination of training size and alpha value were plotted, providing a visual representation of the model's performance across different configurations.

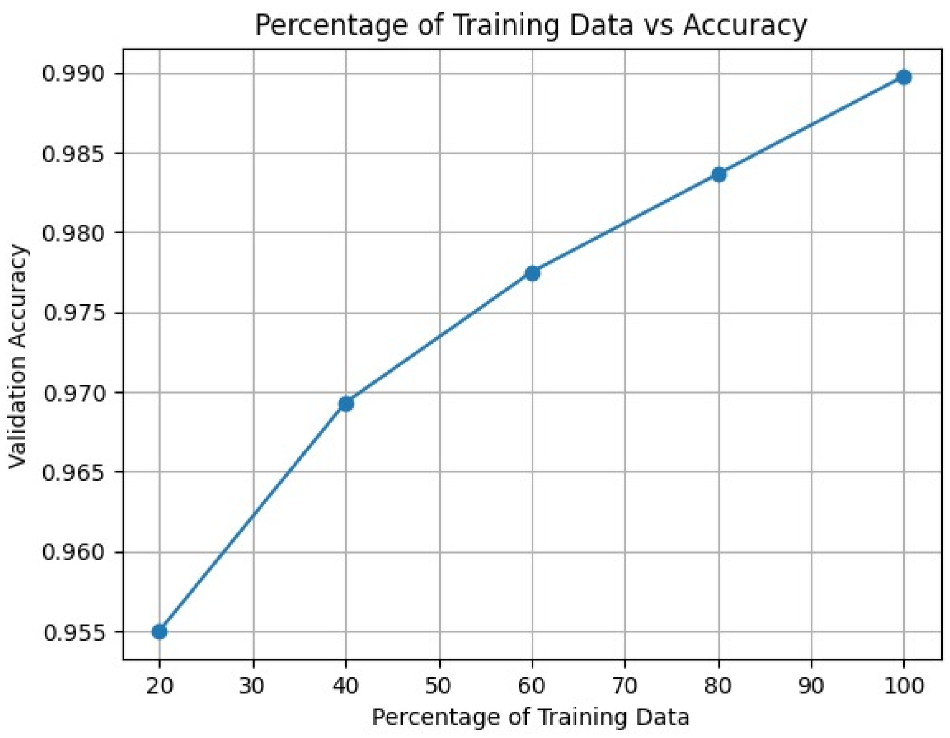
3.3 Model Evaluation

The evaluation of both models involved analyzing the accuracy metrics obtained from the validation set.

3.3.1 Neural Network Model Evaluation

The evaluation of the neural network model focused on the validation accuracy across different percentages of training data. The model's accuracy fluctuated based on the amount of training data used, with the highest accuracy achieved when **100%** of the training data was utilized.

The graph displaying validation accuracy vs. percentage of training data will be inserted here:

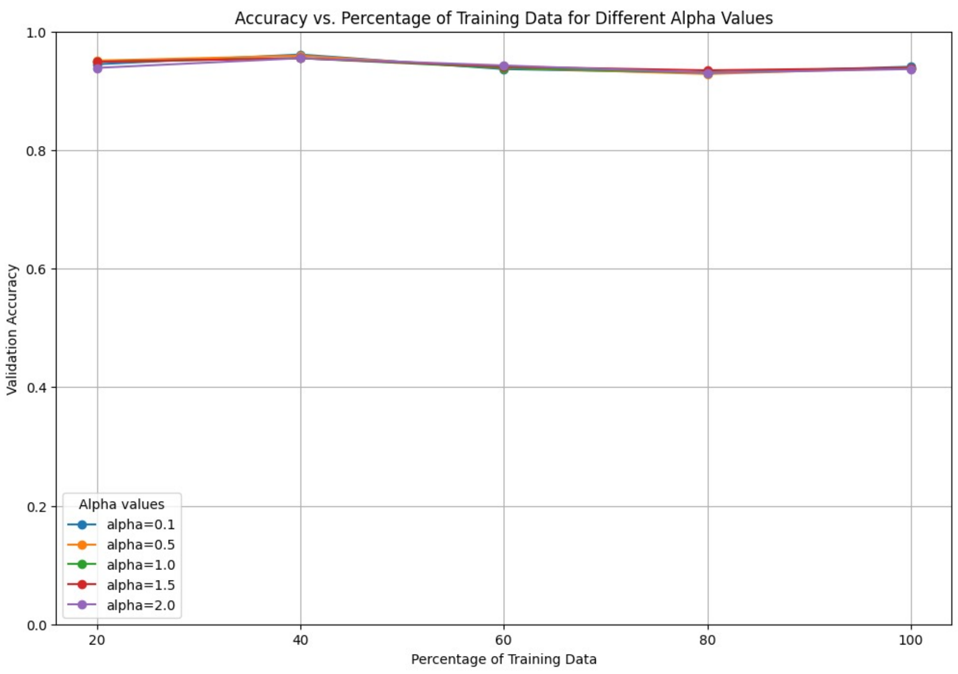


This graph illustrates how the neural network model's accuracy varied with different training data percentages, providing insights into its performance and effectiveness.

3.3.2 Multinomial Naive Bayes Model Evaluation

The evaluation of the Multinomial Naive Bayes model involved analyzing the accuracy metrics obtained from the validation set. The model exhibited stable performance across various alpha values. The validation accuracy results indicated that for **alpha = 0.1**, **40%** of the training data yielded the highest accuracy, further demonstrating the effectiveness of this configuration.

The graph displaying accuracy vs. training data size for various alpha values will be inserted here:



This graph illustrates how the Multinomial Naive Bayes model's accuracy varied with different alpha values and training data percentages, providing insights into the model's robustness and generalization capabilities.

3.4 Best Model Selection

After evaluating the model's performance across various configurations, the best model was selected based on the validation accuracy and Time of Compilation.

- The final chosen model utilized **Multinomial Naive Bayes** with **alpha = 0.1**.
- This configuration yielded satisfactory results, particularly when trained with **40%** of the training data, achieving a good balance between accuracy and efficiency.

The selected model was then applied to the test dataset, and the predictions were saved for further analysis. The validation accuracy with the chosen parameters confirmed the model's effectiveness in predicting binary outputs, establishing a reliable approach for future applications in similar tasks.