Task 1

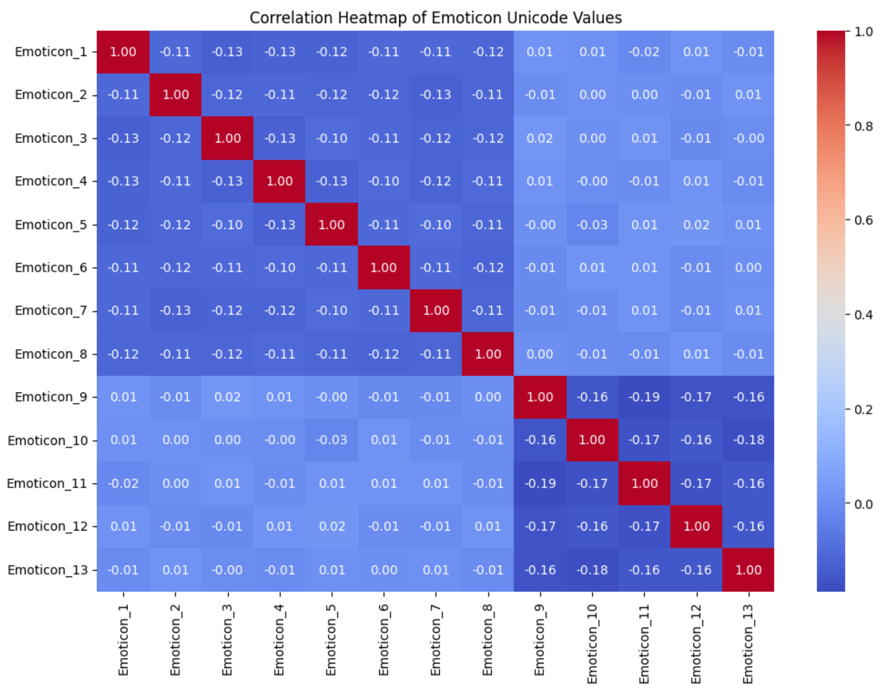
Dataset 1: Emoticons as Features dataset

Dataset Overview

The dataset consists of emoticon-based inputs, where each input is represented by a sequence of Unicode values corresponding to 13 emoticons. These emoticons serve as features of the dataset. We used one-hot encoding to create binary features for each Unicode value, ensuring that the model treats each unique emoticon representation distinctly.

1.1.1 Data Analysis and Feature Transformation

Data Visualization



	Emoticon_1	Emoticon_2	Emoticon_3	Emoticon_4	Emoticon_5	Emoticon_6	Emoticon_7	Emoticon_8	Emoticon_9	Emoticon_10	Emoticon_11	Emoticon_12	Emoticon_13
Emoticon_1	1.00	-0.11	-0.13	-0.13	-0.12	-0.11	-0.11	-0.12	0.01	0.01	-0.02	0.01	-0.01
Emoticon_2	-0.11	1.00	-0.12	-0.11	-0.12	-0.12	-0.13	-0.11	-0.01	0.00	0.00	-0.01	0.01
Emoticon_3	-0.13	-0.12	1.00	-0.13	-0.10	-0.11	-0.12	-0.12	0.02	0.00	0.01	-0.01	-0.00
Emoticon_4	-0.13	-0.11	-0.13	1.00	-0.13	-0.10	-0.12	-0.11	0.01	-0.00	-0.01	0.01	-0.01
Emoticon_5	-0.12	-0.12	-0.10	-0.13	1.00	-0.11	-0.10	-0.11	-0.00	-0.03	0.01	0.02	0.01
Emoticon_6	-0.11	-0.12	-0.11	-0.10	-0.11	1.00	-0.11	-0.12	-0.01	0.01	0.01	-0.01	0.00
Emoticon_7	-0.11	-0.13	-0.12	-0.12	-0.10	-0.11	1.00	-0.11	-0.01	-0.01	0.01	-0.01	0.01
Emoticon_8	-0.12	-0.11	-0.12	-0.11	-0.11	-0.12	-0.11	1.00	0.00	-0.01	-0.01	0.01	-0.01
Emoticon_9	0.01	-0.01	0.02	0.01	-0.00	-0.01	-0.01	0.00	1.00	-0.16	-0.19	-0.17	-0.16
Emoticon_10	0.01	0.00	0.00	-0.00	-0.03	0.01	-0.01	-0.01	-0.16	1.00	-0.17	-0.16	-0.18
Emoticon_11	-0.02	0.00	0.01	-0.01	0.01	0.01	0.01	-0.01	-0.19	-0.17	1.00	-0.17	-0.16
Emoticon_12	0.01	-0.01	-0.01	0.01	0.02	-0.01	-0.01	0.01	-0.17	-0.16	-0.17	1.00	-0.16
Emoticon_13	-0.01	0.01	-0.00	-0.01	0.01	0.00	0.01	-0.01	-0.16	-0.18	-0.16	-0.16	1.00

Figure 1.1: Dataset 1 Feature Distribution
This is a heatmap of the correlation matrix that summarizes the relationships between the 13 features. From the plot, it appears most features have very low correlations (close to zero), indicating they are likely independent of one another.

1.1.2 Experimented Models

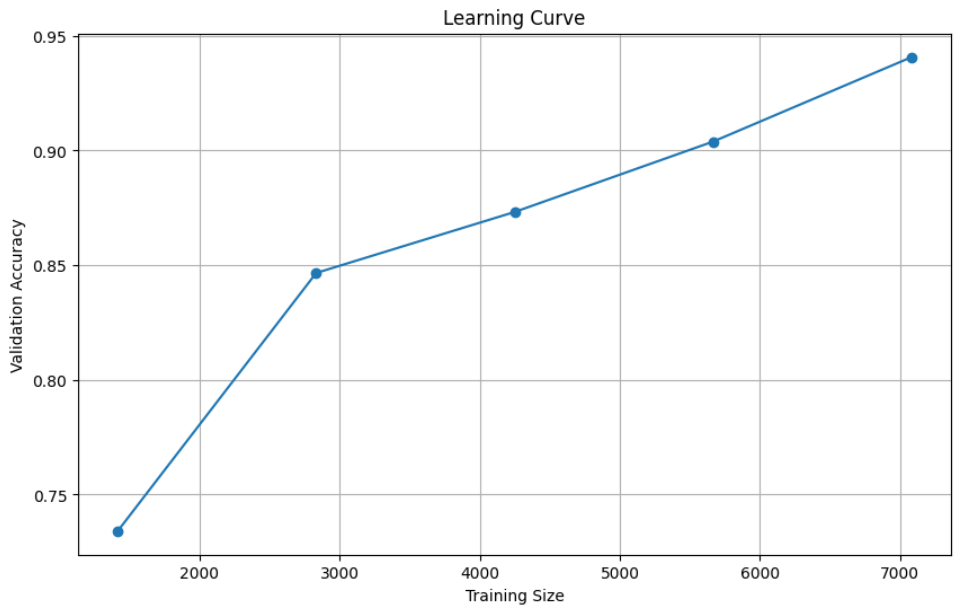
- **Simplicity vs Complexity**: In considering simpler models, we evaluated Logistic Regression. Logistic Regression, while interpretable and straightforward, would struggle with the non-linearities and high dimensionality of our emoticon dataset. Its lack of flexibility makes it less suitable for capturing complex relationships inherent in the data.

- **Ensemble Power**: We also examined Decision Trees and Random Forest methods. Although Decision Trees can capture complex patterns, they are prone to overfitting. In contrast, XGBoost, an ensemble of trees, mitigates overfitting by averaging the decisions of multiple trees, thereby enhancing generalization. While Random Forest serves as another ensemble technique, it lacks the gradient-boosting aspect of XGBoost, leading to a higher bias and reduced effectiveness in capturing residual errors.

- **Optimization**: Finally, we assessed K-Nearest Neighbors (KNN) as an option. KNN is heavily dependent on distance metrics, which become less effective in high-dimensional settings due to the curse of dimensionality. This limitation makes KNN an unsuitable choice for our dataset. In contrast, XGBoost excels in high-dimensional scenarios by optimizing error reduction through gradient boosting, making it the preferred model for our analysis.

1.1.3 Final Model: XGBoost

After encoding the features, we utilized the XGBoost algorithm to train our model. We chose XGBoost due to its efficiency and effectiveness in handling structured data, as well as its ability to capture complex patterns. Recognizing the importance of robust evaluation, we incorporated cross-validation techniques to ensure the model's reliability and to mitigate overfitting.
We employed stratified K-Fold cross-validation, which allowed us to evaluate the model's performance across various data splits. This approach ensured that each fold maintained the same class distribution as the original dataset, providing a more accurate assessment of the model's generalization capabilities.



Results for different training data fractions:

fraction	validation accuracy
0.2	0.734151
0.4	0.846626
0.6	0.873211
0.8	0.903885
1.0	0.940695

Dataset 2: Deep Features Dataset

Dataset Overview

The dataset contains inputs represented as 13×786 matrices, where each of the 13 rows corresponds to a 786-dimensional embedding of an emoticon feature. These embeddings capture the semantic relationships between emoticons, offering a high-dimensional representation of emoticon-based inputs for analysis and classification.

1.2.1 Data Analysis

Data Visualization

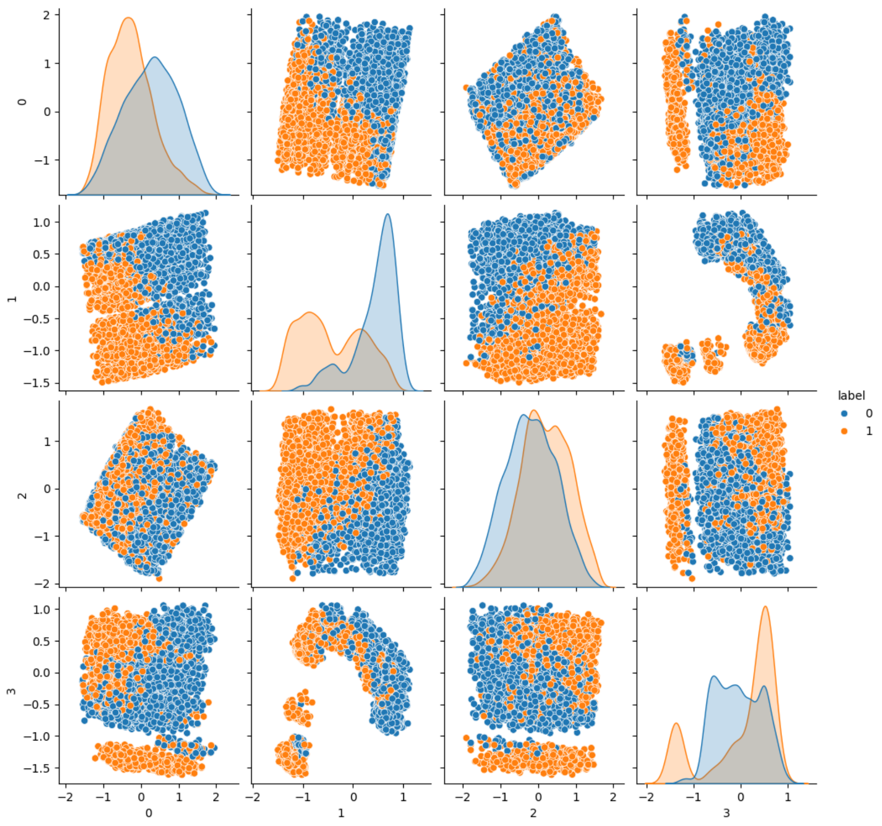


Figure 1.2: Dataset 2 Feature Distribution
The provided pair plot visualizes pairwise relationships between features (768, 769, 770, and 771) in the dataset, with points colored according to two classes (0 and 1). The diagonal plots reveal similar distributions across the classes, although features 770 and 771 show slightly more distinct differences. The scatter plots indicate a mix of points for both classes, suggesting challenges in achieving linear separation. The lack of strong correlations between feature pairs implies that simple linear classifiers may struggle, highlighting the potential need for more complex models like decision trees or neural networks. Additionally, feature engineering or extraction techniques may be essential for enhancing model performance.

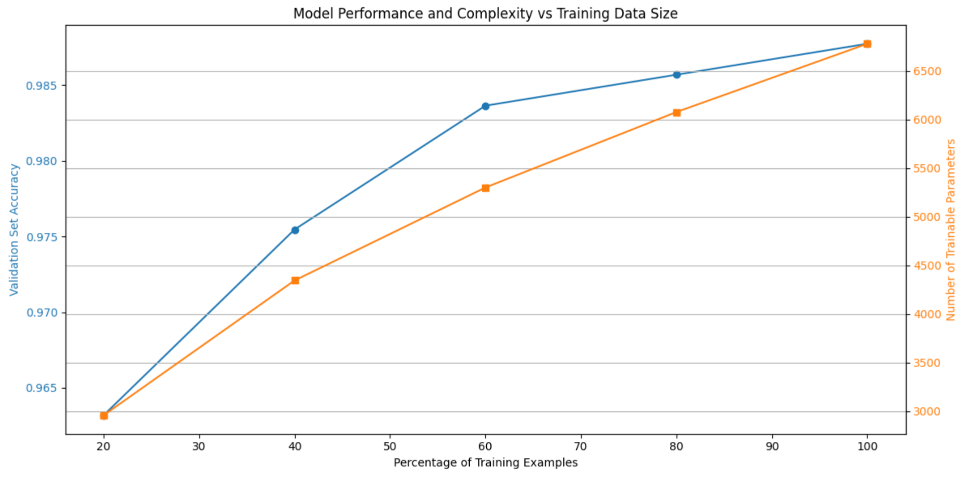
1.2.2 Experimented Models

- **Logistic Regression** offers interpretability and simplicity, but it struggles with the non-linear relationships and high dimensionality present in our emoticon dataset. Given that our features are represented as a 13x786 matrix, the linear assumptions of Logistic Regression may lead to suboptimal performance in capturing the complex patterns inherent in the data.

- **SVM** can model non-linearities by utilizing various kernel functions, providing a more flexible approach to classification. However, SVM is often computationally intensive, particularly in high-dimensional spaces. The model's complexity can lead to long training times and increased resource consumption, diminishing its practicality for our emoticon dataset, which contains many features derived from the embeddings.

1.2.3 Final Model: XGBoost Classifier

Data Preprocessing: we load the features extracted from a deep neural network, represented as 13x786 matrices, and flatten them into vectors of size 9984. Further, we remove constant columns that offer no variance, enhancing computational efficiency by reducing dimensionality. Principal Component Analysis (PCA) reduces the feature space while retaining high variance, mitigating the dimensionality and potentially improving model performance by eliminating noise and redundancy in the data.
Model Training and Tuning: XGBoost classifier proved to be the most effective model for this dataset due to its robust performance in handling high-dimensional data. XGBoost is known for its gradient-boosting framework that builds an ensemble of decision trees, allowing it to capture complex relationships in the data efficiently.
In our implementation, we set specific hyperparameters to optimize model performance. For instance, we set colsample by tree to 0.8, meaning that 80 percent of the features would be randomly sampled for each tree, promoting diversity among the trees and enhancing model robustness.
The learning rate was set to 0.2, which controls the contribution of each tree to the final model, balancing the convergence speed with the risk of overfitting.
The max depth parameter was set to 5, limiting the depth of each tree to prevent overfitting while allowing the model to learn from complex patterns.
Additionally, we used n estimators set to 300, indicating that the model would build 300 trees, providing sufficient complexity to capture the dataset's nuances without excessive overfitting.
This combination of hyperparameters helped balance bias and variance, leading to solid validation performance as reflected in our accuracy evaluations.



fraction	validation accuracy
0.2	0.963190
0.4	0.975460
0.6	0.983640
0.8	0.985685
1.0	0.987730

Table 1.1: Training Fractions and Validation Accuracies

Dataset 3: String-based Features

Dataset Overview

This dataset consists of a collection of inputs represented as strings of length 50, each containing 50 digits. Each digit in the string may correspond to specific features associated with emoticons and deep learning representations derived from the same input found in the other two datasets. However, for the modeling process, we are restricted to utilizing only this dataset, allowing for feature transformations to enhance its effectiveness.

Key Characteristics:

1. **Format:** Each input is represented as a 50-character string, where each character is a digit (0-9). This format necessitates specific feature engineering techniques to derive meaningful representations for model training.

2. **Complexity:** This dataset is considered the most challenging among the three available datasets. The reliance on a string-based representation introduces complexities in understanding and interpreting the underlying relationships between the features.

3. **Relation to Other Datasets:** The string-based features are directly related to emoticon and deep feature representations from the other datasets. While insights gained from those datasets may inform feature transformation techniques, the model training must be exclusively based on the data provided here.

4. **Transformations:** Given the nature of the dataset, applying various feature transformation techniques (such as one-hot encoding, numerical scaling, or embedding methods) can potentially improve model performance. The choice of transformations will be crucial in effectively capturing the underlying patterns inherent in the data.

1.3.1 Data Analysis

Data Visualization

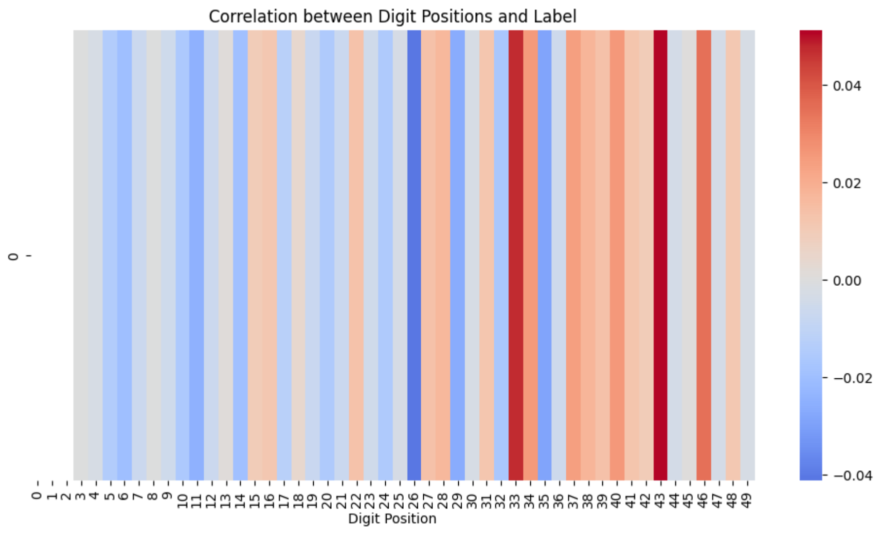


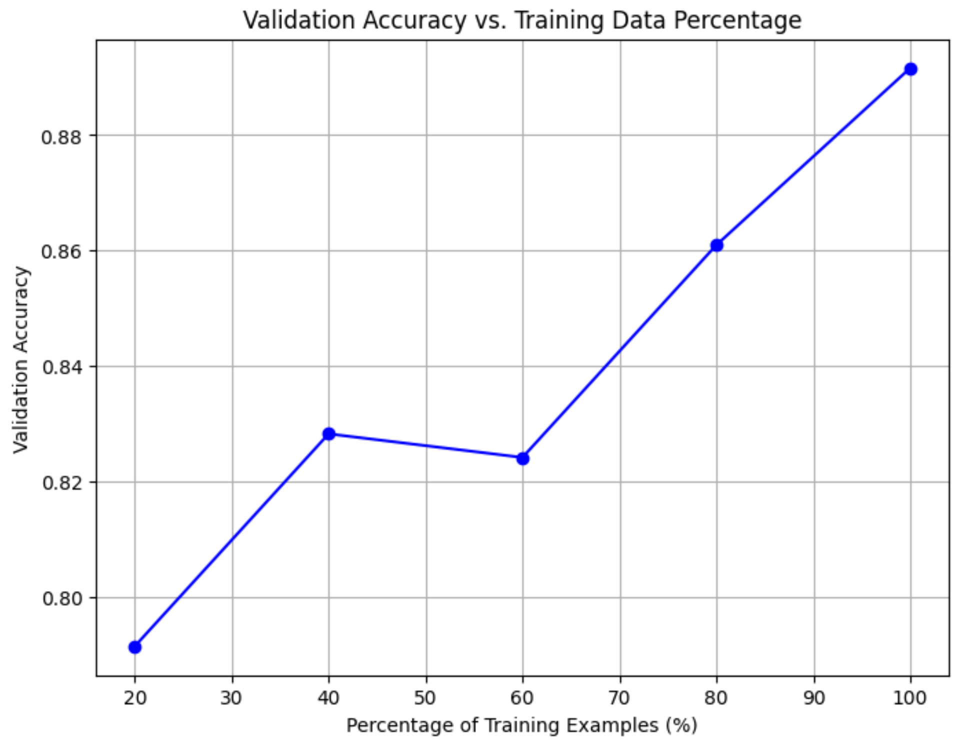
Figure 1.3: Dataset 3 Feature Distribution
Analysis of the 50-digit sequence dataset reveals key structural patterns in its encoded emoticon and deep features. Strong positive correlations emerge at positions 32 and 42 (0.04, 0.035), while early positions (0-10) show consistent negative correlations (-0.03). A distinctive three-position alternating pattern occurs throughout the sequence, with correlations ranging from -0.04 to 0.04. This systematic encoding preserves predictive relationships despite dimensional reduction, suggesting that effective models should leverage both positional information and pattern recognition while giving special attention to strongly correlated positions.

1.3.2 CNN : Convolution Neural Network

Feature Analysis and Transformation: The string sequences are transformed into numerical formats while preserving positional information. Position-wise correlation analysis identifies which digit positions carry more predictive power, aiding in feature selection and enhancing model performance.
Feature Engineering and Pattern Extraction: N-gram analysis (2-grams and 3-grams) captures local patterns within the digit sequences, revealing hidden relationships from the encoded features. CountVectorizer is employed for efficient n-gram extraction, while NumPy facilitates fast computation of position-wise correlations.

The convolutional neural network (CNN) model we developed showed significant improvements in predictive performance for the dataset, thanks to the careful selection of hyperparameters and architectural choices.
Architecture Design: The model architecture consists of two convolutional layers followed by max-pooling layers. The first convolutional layer uses 4 filters with a kernel size of 2, which allows it to capture local patterns in the input sequences effectively. The second convolutional layer increases the number of filters to 8, enhancing the model's ability to learn more complex features. This layered structure is crucial for identifying

overlapping trigrams and extracting spatial hierarchies from the data.
Hyperparameter Optimization: Several hyperparameters were optimized to improve model performance:
Filters: The number of filters in the first and second convolutional layers was set to 4 and 8, respectively. This gradual increase helps the model learn a more complex data representation.
Kernel Size: A kernel size of 2 was chosen for both convolutional layers, allowing the model to focus on pairs of characters (trigrams) effectively.
Dropout Rate: A dropout rate of 0.5 was applied after the dense layer to prevent overfitting, essential for maintaining generalization to unseen data.
Batch Size: The model was trained with a batch size of 64, providing a balance between training speed and model convergence. Epochs: The model was trained for 18 epochs, which was sufficient for the model to learn the underlying patterns without overfitting.
In summary, the specific hyperparameters—such as the number of filters, kernel sizes, dropout rate, batch size, and epochs—were pivotal in developing a robust CNN model that outperformed simpler architectures, leading to improved accuracy and validation metrics for the emoticon sequences dataset.



fraction	validation accuracy
0.2	0.7914
0.4	0.8282
0.6	0.8241
0.8	0.8609
1.0	0.8619

Table 1.2: Training Fractions and Validation Accuracies

Task 2:Model Integration Strategy

Model Combination Approach

In this task, we employ an **ensemble learning strategy** to combine the predictions from three different models trained on distinct feature representations of the same dataset. This strategy leverages the strength of each model to create a more robust final prediction, which is often more accurate than any individual model.

Strategy Details

1. **Stacking Predictions:**
    - We first train three separate models (e.g., Random Forest, Logistic Regression, etc.) on three different feature representations of the same input data. Each model produces predictions (outputs) for both the training and validation datasets.
    - These predictions are then stacked together to form a new dataset. Each row in this new dataset corresponds to the predictions of the three models for a specific input sample.
2. **Weighted Combination:**
    - To reflect the performance of each model, we assign weights to their predictions based on their accuracies on a validation set. This means better-performing models have a larger influence on the final prediction.
3. **Training a Meta-Model:**
    - We train a new model (often called a meta-model) using the stacked predictions as inputs. This meta-model learns to combine the outputs of the base models to improve the overall performance.
    - For instance, we can use a classifier like a Random Forest or a logistic regression model as the meta-model, which takes the weighted predictions as features and outputs the final classification result.
4. **Experimenting with Training Set Sizes:**
    - We will explore different proportions of the training data (20%, 40%, 60%, 80%, 100%) to evaluate how the model's performance varies with the amount of training data used. This will help identify the optimal size for training the meta-model.

This ensemble approach allows us to leverage the unique strengths of each model trained on the different feature sets, potentially leading to superior performance compared to individual models.

Final Model Result

: Initially, the accuracies of three previously trained models are 0.94, 0.98, and 0.86, respectively. These accuracies are then converted into a weight vector, where each model's influence on the final predictions is proportional to its accuracy. The weights are normalized to ensure they sum to one, allowing for balanced contributions from each model when creating the combined training and validation datasets.

The predictions from the three models are stacked to form the training dataset (X train) and the validation dataset (X val). This stacking is achieved through the np. stack function, which consolidates the predictions from each model into a structured format. Subsequently, the stacked predictions are weighted using the previously computed normalized weights to create X train weighted and X valweighted. This weighting step ensures that more accurate model predictions have a greater impact on the final output. The RandomForestClassifier is then initialized and trained on the weighted training data using the fit method, with the labels for the training set provided by the y train. After training, the model makes predictions on the weighted validation set (X val weighted) using the prediction method. The model's performance is evaluated by calculating the accuracy and balanced accuracy scores using the accuracy score and balanced accuracy score, respectively. These metrics provide insights into the model's effectiveness in making accurate predictions.
Overall, the code effectively combines predictions from different models, utilizing a robust ensemble approach through the Random Forest Classifier and implementing a systematic method of weighting predictions based on prior accuracies. This strategy not only aims to surpass the accuracy of individual models but also sets the stage for further experimentation with varying training set sizes, thereby optimizing the model's performance on the validation dataset.