CS771-Intro to ML(Autumn 2024): Mini-project 1-Report

Group 95 - Cerebro

Aman Yadav(220121),Anirvan Tyagi(230143),Kishore S(230566),Varsha Pillai(231121)

Contents

1 Abstract

This mini-project addresses binary classification using three datasets derived from the same raw data but featuring different input representations: emoticons, deep features, and digit sequences. The goal is to develop models for each dataset individually, optimizing both accuracy and training data usage, while adhering to a 10,000 parameter limit. Additionally, we explore whether combining the datasets through feature merging can enhance model performance. By varying training data sizes and evaluating validation accuracy, the project aims to identify the best-performing models and assess the potential benefits of combining multiple feature representations for improved generalization to unseen data.

2 Datasets Analysis

The dataset consists of three distinct representations: emoticons, deep features, and digit sequences. The emoticon dataset includes inputs of 13 characters each, with binary labels. The deep features dataset represents inputs as a 13×768 matrix, capturing higher-level abstractions. Lastly, the digit sequence dataset encodes inputs as sequences of 50 digits. Each dataset contains corresponding training, validation, and test sets, providing different feature representations for the same underlying data.

3 Emoticon Dataset

3.1 Methodology

This dataset consists of 13 categorical features representing emoticons. To handle this discrete data, models like Logistic Regression, Decision Trees, and Random Forests were initially considered for their simplicity. However, to achieve higher accuracy, the data was first pre-processed using an LSTM-based neural network to extract useful features from the emoticon sequences. Traditional classifiers such as KNN, Decision Tree, Random Forest, and SVM were then trained on these extracted features to identify the best-performing model.

3.1.1 LSTM-Based Feature Extraction

Neural Network Architecture:

- Embedding Layer: The emoticon sequences were tokenized at the character level and passed through an embedding layer, where each unique character was mapped to an 8-dimensional space.
- LSTM Layer: A 16-unit LSTM layer was employed to capture the temporal dependencies in the emoticon sequences.
- Dense Layers: An 8-unit dense layer with ReLU activation was added for feature extraction, followed by a single output neuron with a sigmoid activation for binary classification.

Training Details:

- Optimizer: Adam optimizer was used to minimize the binary cross-entropy loss.
- Epochs: The model was trained for 10 epochs with a batch size of 32. Validation: Validation data was used to monitor the model's performance.

Feature Extraction: After training the LSTM-based network, the features from the penultimate dense layer were extracted. These features served as the input for traditional classifiers.

3.1.2 Model Training and Evaluation

Classifiers: After extracting features, four traditional classifiers were trained:

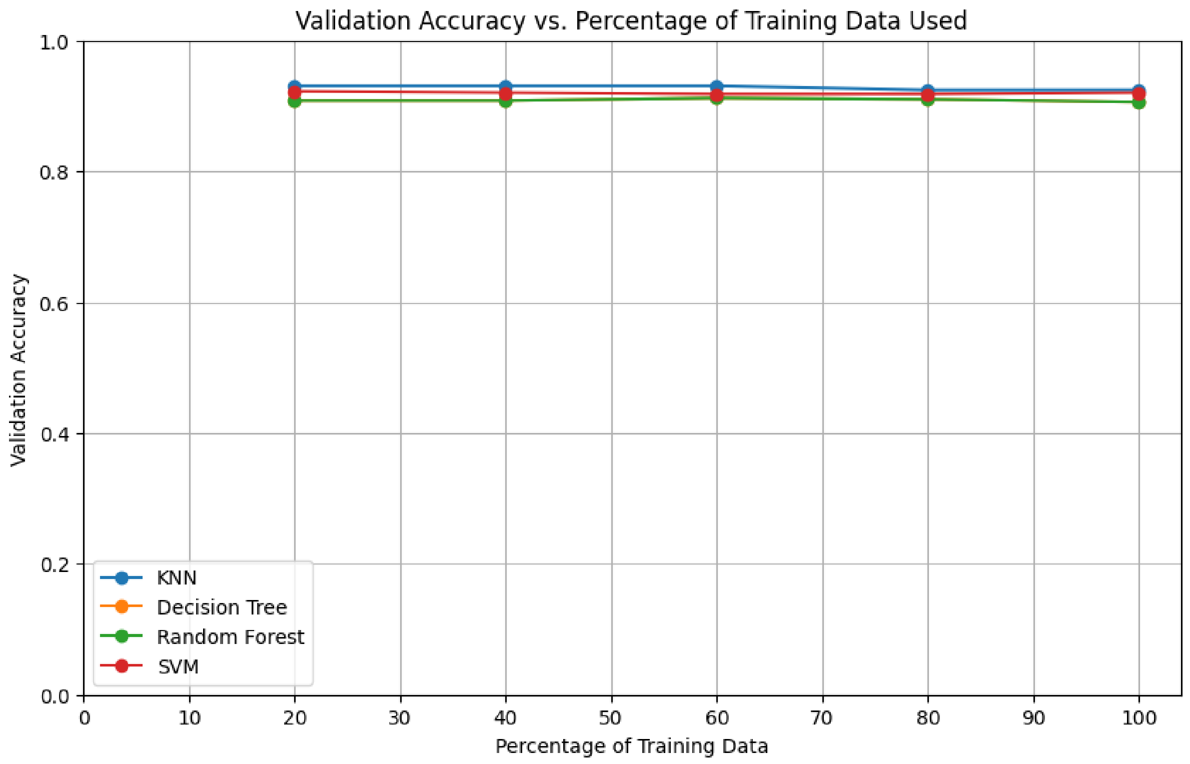
- KNN (K-Nearest Neighbors): A proximity-based classifier.
- Decision Tree: A tree-based model that splits data based on feature values.
- Random Forest: An ensemble of decision trees used to improve model stability.
- SVM (Support Vector Machine): A linear classifier aiming to maximize the margin between data points.

Training with Different Data Percentages: The classifiers were trained with varying percentages of the training data (20%, 40%, 60%, 80%, 100%) to observe how data size affected performance.

Validation Accuracy: Each classifier was evaluated on the validation dataset after training with different portions of the data. The validation accuracy for each percentage was recorded, and the model performance was compared across classifiers.

3.2 Results

3.2.1 Validation Accuracy vs % Training Data Plot



3.2.2 Classifier Performance

Several classifiers were evaluated using the extracted deep features. Each model was trained on different percentages of the training data, and validation accuracy was recorded as shown below:

- KNN:Achieved the highest validation accuracy of 93.05% with up to 60% of the data and 92.43% when using the full dataset.

- Decision Tree:Validation accuracy peaked at 91.21% with 60% of the data, but performance slightly decreased with more data, reaching 90.59
- Random Forest:Similar performance to the Decision Tree, with a peak at 91.21% using 60% of the data and 90.59% using 100%.
- SVM: SVM reached a maximum validation accuracy of 92.23% and maintained consistent performance between 91.82% and 92.02% across all training sizes.

3.2.3 Best Model Selection

KNN emerged as the best-performing model with a final validation accuracy of 92.43% on the full dataset, making it the most suitable classifier for the emoticon dataset in this task. The Model required 3465 trainable parameters.

3.3 Conclusion

The trained KNN classifier was applied to the test data, and predictions were generated based on the deep features extracted by the LSTM model. The results were saved in a text file ”pred_emoticon.txt” for further evaluation.

4 Deep Features Dataset

4.1 Data preprocessing

4.1.1 Method 1: Averaging

To simplify the input representation, the embeddings are **averaged across the 13 features**, resulting in a 768-dimensional feature vector for each sample. This approach reduces the complexity of the dataset.

But the problem with averaging is that the dataset is structured as a 13×768 matrix, where each row likely represents a specific feature or a meaningful aspect of the input (e.g., different emoticon features). Simply averaging the 13 embeddings collapses all this detailed information into a single 768-dimensional vector. Averaging eliminates the sequential or positional relationships between the rows (the 13 ”steps” or features), which may be crucial for classification. This loss of structure could lead to suboptimal feature representations, affecting the model's ability to learn intricate patterns. So I tried another preprocessing method known as **LSTM**.

4.1.2 Method 2: LSTM preprocessing

LSTM stands for Long Short-Term Memory. It is a type of recurrent neural network (RNN) architecture that is specifically designed to model sequential data and capture long-range dependencies. LSTM works on 3D input data of the following shape: (samples, time steps, features).When we preprocess the data using LSTM, we're using it to extract higher-level features from this sequential input.

Preprocessing the given data using LSTM: The input is already in a format that can work well with LSTM since it's a 2D matrix for each sample: (13, 768). In LSTM, the ”time steps” correspond to 13 emoticon features, and the ”features” correspond to the 768-dimensional embeddings. The LSTM will learn to capture the dependencies and relationships between the 13 emoticon features for each input. After processing, the LSTM can return a fixed-length representation (such as the final hidden state) that summarizes the information in the sequence. Once the LSTM processes the sequence, the output can be fed into models (like logistic regression, SVM, etc.) for classification.

The code for preprocessing is as follows:

```
model = Sequential()
model.add(LSTM(64, input_shape=(X_train.shape[1], X_train.shape[2])))
```

Output of LSTM : Each input sample of shape 13×768 is reduced to a single output vector of size 64 after processing through the LSTM.

4.2 Methodology

The following machine learning models were evaluated for binary classification : Support Vector Machine (SVM), Decision Tree, Logistic Regression, Random forest. In addition to traditional machine learning models, I also implemented a custom prototype-based model, *Learning with Prototype (LwP)*, to evaluate its effectiveness compared to standard classifiers. To evaluate the models, experiments were conducted by varying the amount of training data used and measuring how the performance changes on the validation set. The percentage of training data used was varied across five levels: **20%, 40%, 60%, 80%, and 100%**.

For each percentage of training data: A subset of the training data was selected based on the percentage (e.g., for 20%, the first 20% of the training examples were used). Each model was trained using the selected training subset. The trained models were used to predict the labels of the validation set. The validation accuracy was calculated by comparing the predicted labels to the true labels of the validation set.

This process was repeated for each model and for each percentage of training data, allowing us to observe how the validation accuracy changes as more training data is used.

4.3 Plotting the results

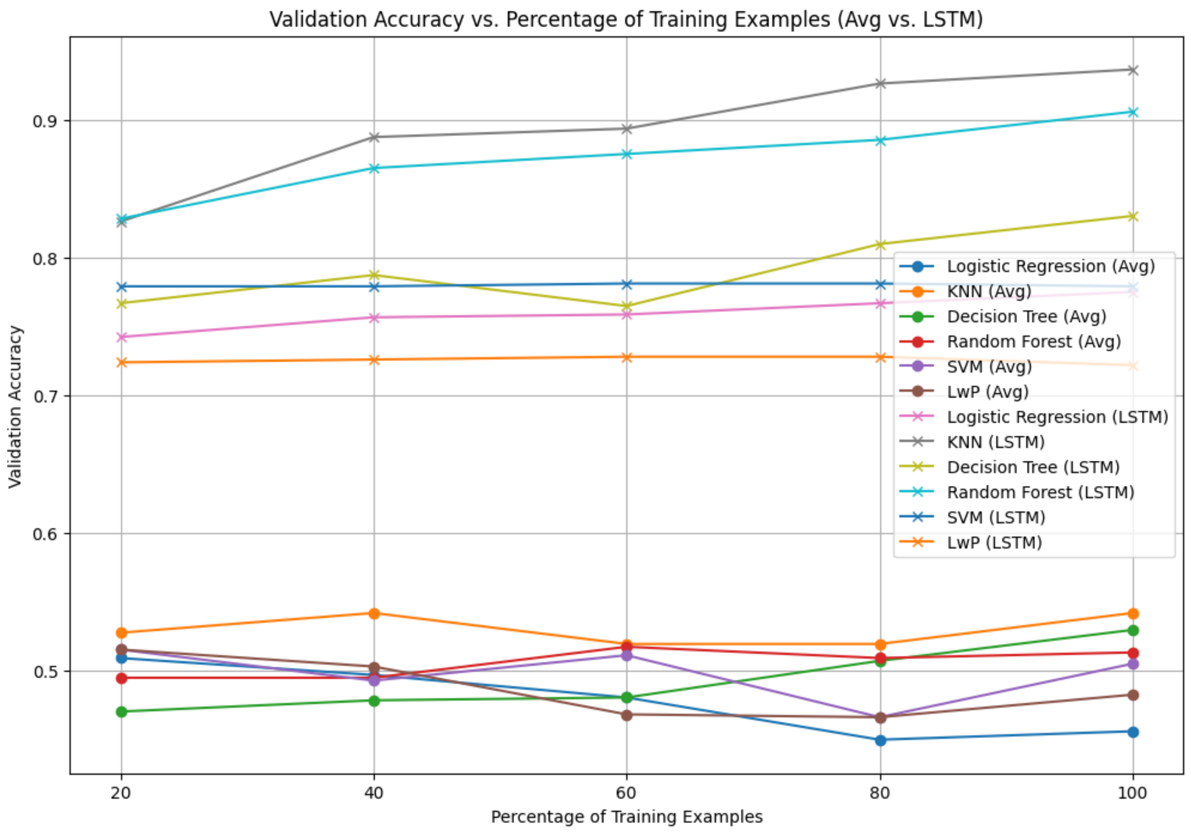


Figure 1: Validation Accuracy vs. Percentage of Training Examples for Different Classifiers

A plot was generated to show the **validation accuracy** of both the methods of preprocessing i.e., averaging and LSTM. The **X-axis** represents the percentage of training data, and the **Y-axis** represents the validation accuracy. Each model is represented by a separate line on the plot.

This plot helps visually compare the performance of the models across different training data sizes and assists in identifying the model that balances high accuracy and minimal data usage.

4.4 Conclusion

Based on the two plots, comparing the validation accuracy with respect to the percentage of training examples for both the **averaging approach** and **LSTM preprocessing**, here are the key observations and conclusions:

4.4.1 Averaging Approach:

The highest accuracy across models is around **0.54**, with most models struggling to perform better than random guessing (50%). Most models (Logistic Regression, KNN, Decision Tree, etc.) show **inconsistent performance** as the amount of training data increases.For example, KNN peaks at around 60% training data and then decreases, while Logistic Regression declines significantly from 40% to 100%.This method does not provide reliable improvements in validation accuracy, and the performance seems unstable across different training sizes and models.

The results suggest that averaging does not efficiently capture meaningful patterns from the features, leading to poor generalization across models.

4.4.2 LSTM Preprocessing :

There is a substantial improvement, with the highest accuracy reaching approximately **0.92-0.94**. All models perform significantly better compared to the averaging approach. LSTM preprocessing provides a dramatic improvement in accuracy and leads to more consistent and stable performance. This indicates that LSTM is effectively capturing sequential dependencies and providing better feature representations for classification tasks.

Best Model: KNN stands out as the best model with LSTM preprocessing, achieving the highest validation accuracy (around 0.92-0.94).

5 Text Sequence Dataset

5.1 Methodology

1. **Data Preparation:**
   - **Data Cleaning**: The dataset consisted of digit sequences as input features. These sequences were cleaned by removing non-digit characters, and the cleaned sequences were transformed into integer arrays.
   - **Padding**: To handle the varying lengths of sequences, padding was applied to ensure uniform input dimensions.
   - **One-Hot Encoding**: The padded sequences were one-hot encoded to create a feature matrix suitable for machine learning models.
2. **Models Tested**:
   - **Logistic Regression**: A classification model that uses a linear decision boundary.
   - **Random Forest Classifier**: An ensemble of decision trees that works by averaging multiple trees to improve predictive accuracy.
   - **Decision Tree Classifier**: A model that splits data recursively into subsets based on feature value thresholds.
   - **K-Nearest Neighbors (KNN)**: A distance-based classification model that classifies based on the majority class of the nearest neighbors.

Model	20% Data	40% Data	60% Data	80% Data	100% Data
Logistic Regression	63.27%	66.12%	65.51%	66.53%	65.10%
Random Forest	55.51%	61.63%	63.88%	60.82%	63.47%
Decision Tree	56.12%	54.08%	58.16%	55.92%	59.18%
K-Nearest Neighbors	54.08%	57.14%	56.73%	54.29%	56.12%
SVM	62.86%	64.90%	67.76%	65.92%	67.35%

- **Support Vector Machine (SVM)**: A classifier that separates classes using a hyperplane that maximizes the margin between them.

3. **Evaluation**:
   - The models were trained with **different percentages** of the training data (20%, 40%, 60%, 80%, and 100%) to evaluate how they perform with varying amounts of data.
   - The models were then evaluated on the same validation set, and the **accuracy** was calculated for each model at each percentage of training data.
4. **Visualization**: A plot was generated to compare the accuracies of all models across the various percentages of training data.

5.2 Results

- **Logistic Regression** performed consistently well across different percentages of training data, achieving the highest accuracy at **66.53% with 80% training data**. Its performance dropped slightly when the full dataset was used.
- **SVM** also performed reliably well, achieving its peak accuracy of **67.76% with 60% training data**. It maintained stable accuracy across all percentages.
- **Random Forest** demonstrated fluctuating performance, achieving its highest accuracy of **63.88% with 60% training data**.
- **Decision Tree** showed less consistency, with accuracy peaking at **59.18% with 100% training data** but generally underperforming compared to Logistic Regression and SVM.
- **K-Nearest Neighbors (KNN)** performed the worst overall, with a maximum accuracy of **57.14% with 40% training data**, and its performance was generally unstable as the training data increased.

Summary of Accuracies:

5.3 Conclusion

- **Logistic Regression** and **SVM** emerged as the top-performing models, with SVM showing slightly better accuracy in general. Both models demonstrated the ability to handle the dataset well, especially when a moderate percentage of training data (60%-80%) was used. These models could benefit from further fine-tuning of hyperparameters.
- **Random Forest** showed some potential, though its performance was more variable. Hyperparameter tuning (e.g., increasing the number of trees or adjusting tree depth) could help improve its consistency.
- **Decision Tree** and **KNN** underperformed relative to the other models. These models might benefit from more advanced feature engineering or more specialized hyperparameter optimization.

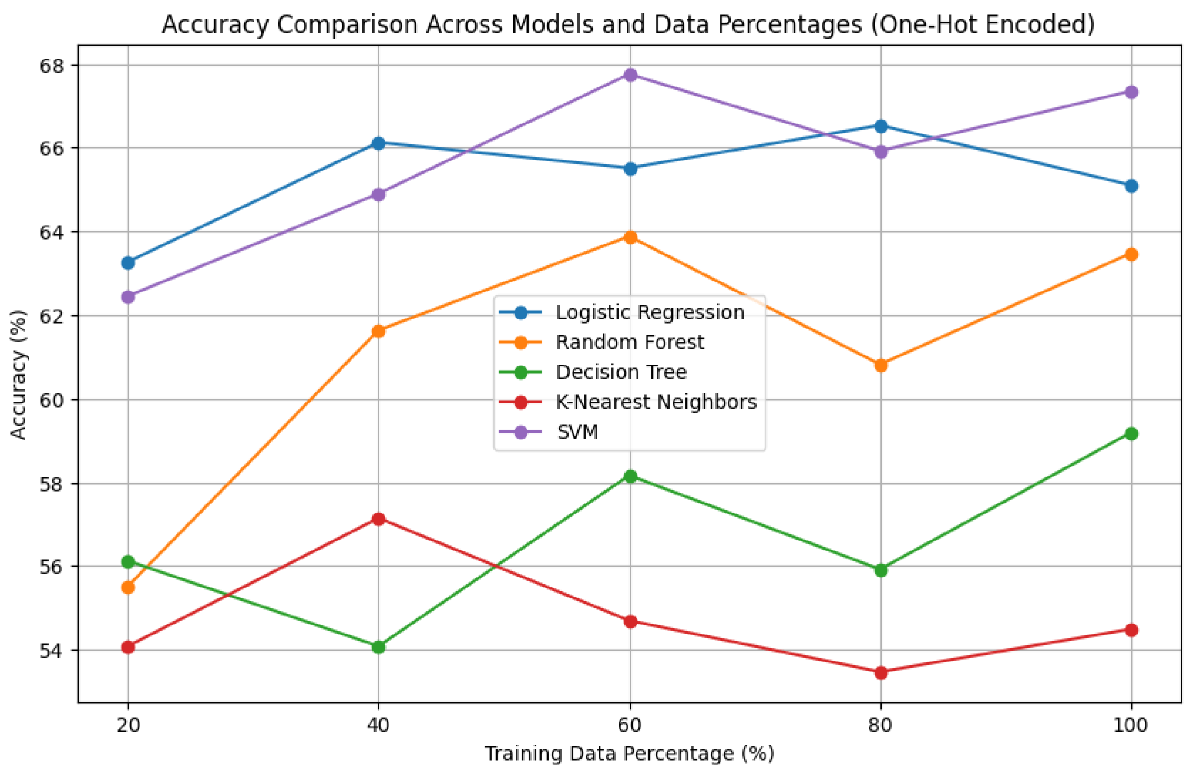


Figure 2: Validation Accuracy vs. Percentage of Training Examples for Different Classifiers

In conclusion, **SVM** and **Logistic Regression** are the most promising candidates for further exploration and optimization, while **Random Forest** may also be worth improving. **KNN** and **Decision Tree** would require significant adjustments to match the performance of the other models in this task. Here is the plot.

6 Combining the Datasets

6.1 Pre-processing the Datasets

In this approach, we combined the three datasets—emoticons, text sequences, and deep features—into a single feature set to train classifiers that leverage all available information.

- Emoticon Dataset: Each input emoticon sequence was first tokenized at the character level using a tokenizer. These tokenized sequences were then padded to ensure uniform input lengths across all examples. The padded sequences were treated as a feature set.
- Text Sequence Dataset: Similar to the emoticon data, the text sequences were tokenized at the word level and then padded to create uniform input lengths. This provided another set of features derived from the sequence representation of the data.
- Deep Feature Dataset: The deep features, which consist of a matrix of size 13×768, were reshaped and flattened to fit into a 2D format. This enabled us to treat each example's deep features as a separate feature vector.

6.2 Combining into a single dataset

- Combining Features: After pre-processing each dataset individually, the features from all three datasets were concatenated horizontally for each corresponding training, validation, and test instance. This resulted in a single, combined feature set that incorporates the information from emoticon sequences, text sequences, and deep features.

- Scaling: Since the combined feature space includes data from different sources (discrete tokenized data and continuous deep features), the concatenated data was scaled using StandardScaler to normalize the feature values, ensuring that each feature type contributes equally to the training process.

By merging the datasets in this manner, we aimed to utilize the strengths of each representation—categorical information from the emoticons, sequential information from the text, and semantic information from the deep features—to train classifiers with richer input data. This combination allowed for a more comprehensive feature set, which we then used to train various machine learning models. The final result was a feature set that could potentially offer better generalization by capturing diverse aspects of the raw data.

6.3 Results

The results of training models on the combined dataset show significant improvements in validation accuracy compared to training on individual datasets.

6.3.1 Validation Accuracy VS % Training Data Plot

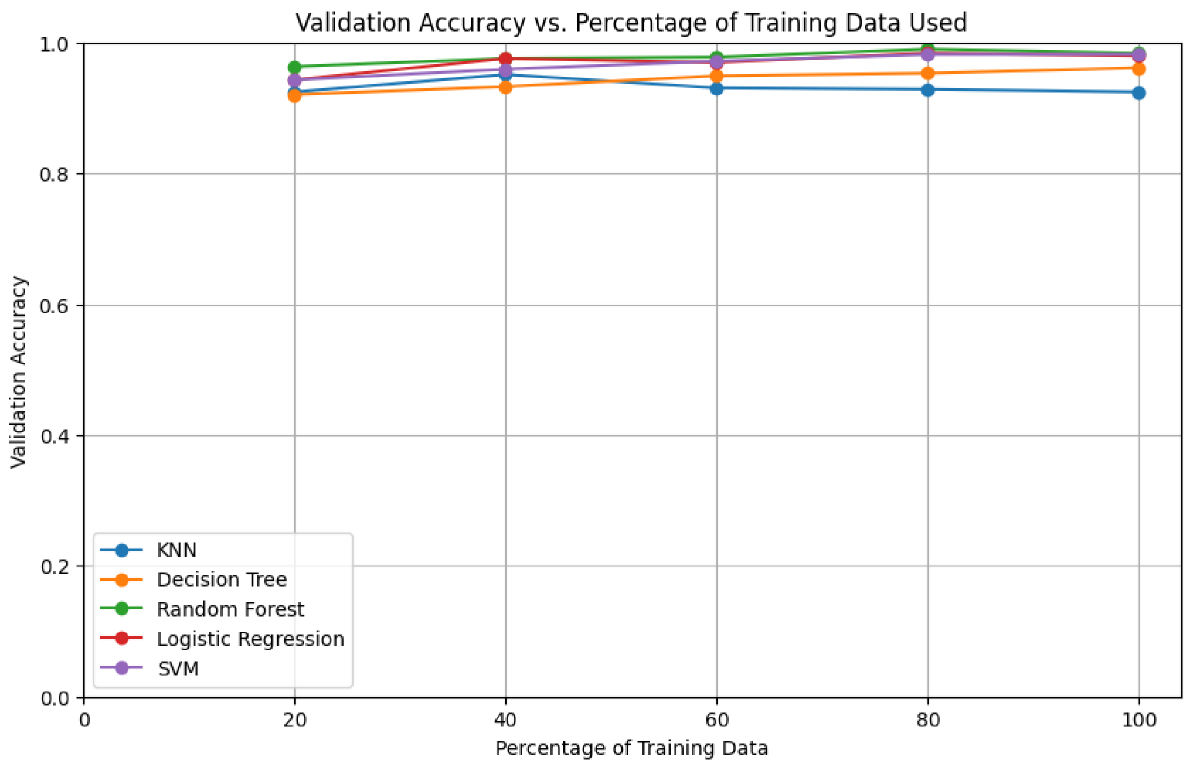


Figure 3: Validation Accuracy vs % Training Data for Combined Dataset

Here's a breakdown of the key insights:

- KNN: Achieved its highest validation accuracy (0.9509) when using 40% of the training data. However, the performance slightly decreased as more data was used, with the final accuracy stabilizing at 0.9243 when using the full dataset.

- Decision Tree: This model performed increasingly well as more training data was introduced, reaching its peak accuracy of 0.9591 when trained on the full dataset. The model showed consistent improvement with additional data.
- Random Forest: This model achieved excellent validation accuracy from the beginning, with 0.9571 at 20% of the data, peaking at 0.9836 using the full dataset. Random Forest was the best-performing model overall, demonstrating its strength in handling the combined feature set by achieving the highest validation accuracy across all models.
- Logistic Regression: This model also performed very well, achieving 0.9755 accuracy with just 40% of the data and reaching a peak of 0.9836 at 80%. Although its performance slightly dropped when using the full dataset, it still remained competitive.
- SVM: Despite some convergence issues, SVM consistently improved as more data was used, reaching 0.9816 accuracy with 80% and 100% of the data. However, it didn't outperform Random Forest in the final comparison.

6.4 Conclusion

The results obtained from the combined dataset clearly outperform the results achieved by models trained on individual datasets. Random Forest emerged as the best model, achieving a validation accuracy of 0.9836, which is higher than any of the models trained on the individual datasets. The performance improvement can be attributed to the diverse and complementary nature of the feature sets—emoticons, text sequences, and deep features—which, when combined, provided a richer representation of the data. This highlights the effectiveness of using feature combinations in improving model performance, allowing for better generalization on unseen test data.

References

[1] Christopher Bishop. *Pattern Recognition and Machine Learning.*

[2] *Neural Networks and Deep Learning.*A Textbook by Charu Aggarwal

[3] Keras; LSTM Documentation
$https://keras.io/api/layers/recurrent_layers/lstm/$

[4] OpenAI. (2024). ChatGPT (Version 3.5)Language model.Retrieved from
$https://chat.openai.com/$