Intro to Machine Learning (CS771)
Mini-Project 1

Ayush Meena
220268

Archit Atrey
220195

Rishikesh Dargad
220891

Patil Piyush Shivaji
220759

Group Name - PaaaR

Group No - 5

Instructor : Dr. Piyush Rai

Introduction

This mini-project focuses on the **training** and **evaluation** of machine learning models for **binary classification** across three distinct datasets derived from the same raw data. Each dataset provides a unique feature representation of the inputs: (1) **emoticons** encoded as categorical features, (2) features **extracted by a deep neural network**, and (3) a **sequence of digits**. We explore different models tailored to each dataset, emphasizing two critical metrics: **classification accuracy** and the **amount of training data** needed for effective model performance. In addition, we investigate whether a **unified model**—combining all three datasets—can surpass the performance of individual models. The project aims to balance accuracy with training efficiency, utilizing a variety of machine learning techniques to **optimize model performance** and provide insights into the trade-offs between feature representations and learning efficiency.

Task 1

In this task, we will perform binary classification on three provided datasets. The objective is to develop models that accurately classify instances into one of two categories based on the given features in each dataset.

Emoticons as Features Dataset :

In this dataset, the features of each input are given in the form of 13 emoticons.
We have to classify each input into one of two labels: 0 or 1.

- **Data Preprocessing :** The following key preprocessing tasks were performed on the datasets -

  - **Extracting Emoticons from Input Strings** : The raw input data consists of emoticons represented as strings. These strings were converted into lists of individual emoticons. This transformation allows the model to treat each emoticon as a distinct element.
  - **Mapping Emoticons to Indices** : To allow the model to process emoticons numerically, each unique emoticon was mapped to an integer index. The mapping process created a dictionary where each emoticon is assigned a unique index starting from 1.

- **Model Development and Evaluation :**

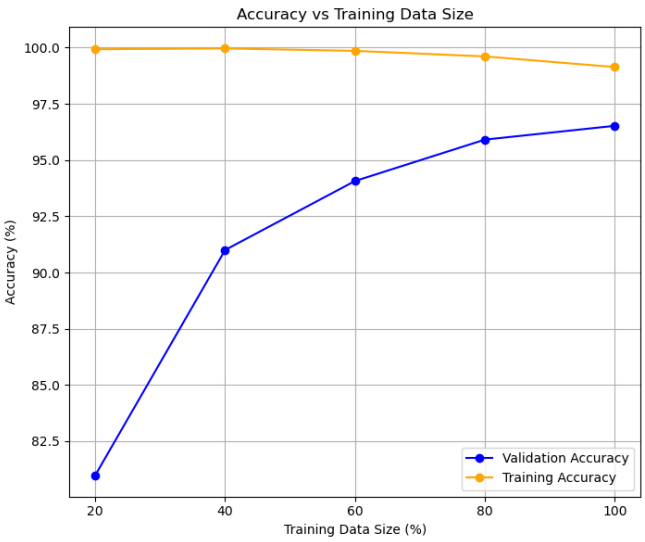
  - **Architecture:** A custom Recurrent Neural Network (RNN) architecture was designed to process the emoticon sequences. The model consists of three key layers:
    - ⋆ **Embedding Layer :** The first layer converts each emoticon (represented by an index) into a dense, continuous vector of size 8. This helps the model better understand the relationship between different emoticons.
    - ⋆ **RNN Layer :** Next, we use a Simple RNN with 16 units. This layer looks at the sequence of emoticons and tries to capture the patterns over time, learning how emoticons relate to each other in the sequence.
    - ⋆ **LSTM Layer :** After the RNN, an LSTM (Long Short-Term Memory) layer with 32 units is used. This layer helps the model remember important patterns from earlier in the sequence, which can help improve its understanding of the overall sequence.
    - ⋆ **Fully Connected Layer :** Finally, a Dense layer with a **sigmoid activation** function is used to make the final prediction (either class 0 or class 1), which represents the binary classification outcome.

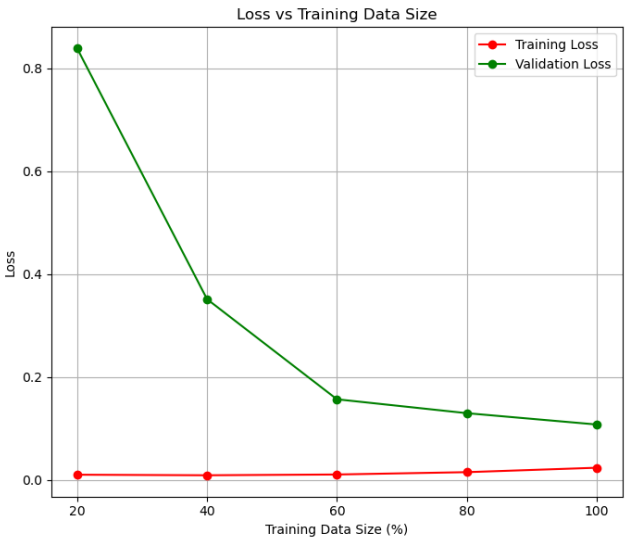
  - **Training :** The model was trained using the preprocessed training data. We used the **Adam optimizer** with a learning rate of 0.001 to help the model learn. The loss function used is **binary cross-entropy**, which is appropriate for the binary classification problem. The model was trained for 20 epochs with a batch size of 32.
  - **Evaluation :** Once trained, the model's performance was tested on a separate validation set. The model was evaluated using **accuracy**, which shows the proportion of correct predictions.
  - **Trainable Parameters :** Total number of trainable parameters $\rightarrow$ **8,425**.

- **Accuracy vs Training Data Size :**

The model's validation accuracy on the validation dataset for different training dataset sizes is as follows:

- **Training size : 20% $\rightarrow$ Validation Accuracy : 86.91%**
- **Training size : 40% $\rightarrow$ Validation Accuracy : 93.05%**
- **Training size : 60% $\rightarrow$ Validation Accuracy : 94.27%**
- **Training size : 80% $\rightarrow$ Validation Accuracy : 95.91%**
- **Training size : 100% $\rightarrow$ Validation Accuracy : 96.73%**

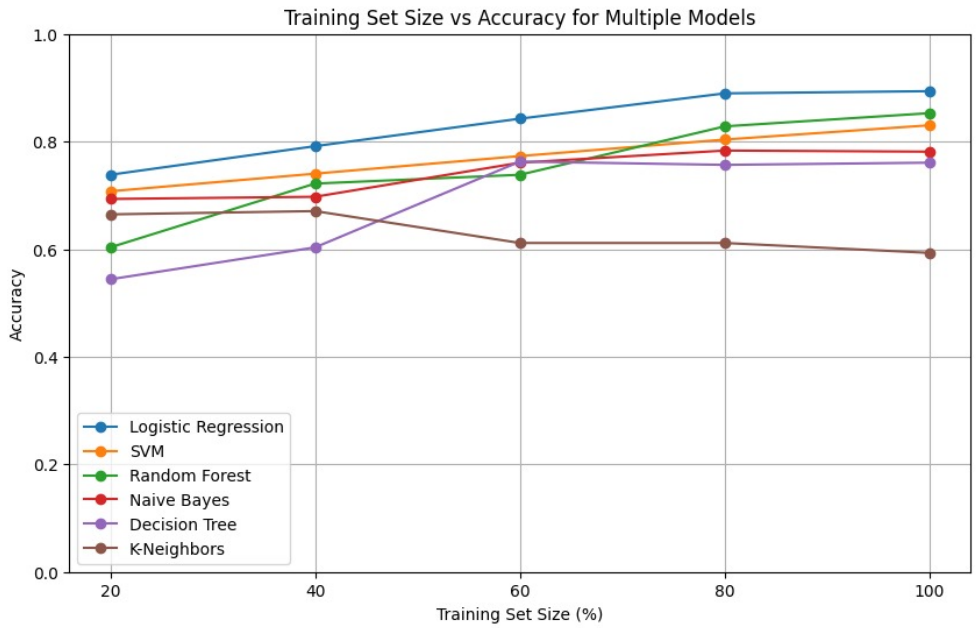




- **Other Models Tried :**

These models were also explored but were ultimately rejected due to lower accuracy compared to the final model:

- **Logistic Regression :** Achieved an accuracy of 89.37%. Logistic regression works well with categorical features and performed reasonably well with properly preprocessed input data.
- **Random Forest Classifier :** Reached an accuracy of 85.28%. Random forest's ensemble approach helps in handling non-linear relationships between features and labels.
- **Support Vector Machine (SVM) :** Attained an accuracy of 84.04%. SVM models are known for their robustness in high-dimensional spaces, making them effective for classification tasks.
- **Naive Bayes Classifier :** Produced an accuracy of 78.12%. The Naive Bayes model assumes feature independence, which may not fully capture the underlying structure of the data.
- **Decision Tree Classifier :** Achieved an accuracy of 76.07%. While decision trees are prone to overfitting, they are interpretable and performed decently on this task.
- **K-Neighbors Classifier :** Scored an accuracy of 59.30%. The K-Neighbors algorithm struggles with high-dimensional data, resulting in lower performance.



Deep Features Dataset :

Each input is represented as a 13×768 matrix, with 768-dimensional embeddings for 13 emoticon features.
The task is to classify the input into one of two labels: 0 or 1.

- **Data Preprocessing :** The data preprocessing steps involved the following key operations to prepare the dataset for model training and evaluation:

- **Flattening:** The input features, originally represented as 13×768 matrices, were flattened into 1D arrays of size 9,984. This step ensures the model can process the features as a vector of values.
- **Feature Scaling:** The **StandardScaler** was applied to normalize the features, scaling them to have a mean of 0 and a standard deviation of 1. This helps in reducing bias due to differing feature magnitudes and accelerates model convergence.

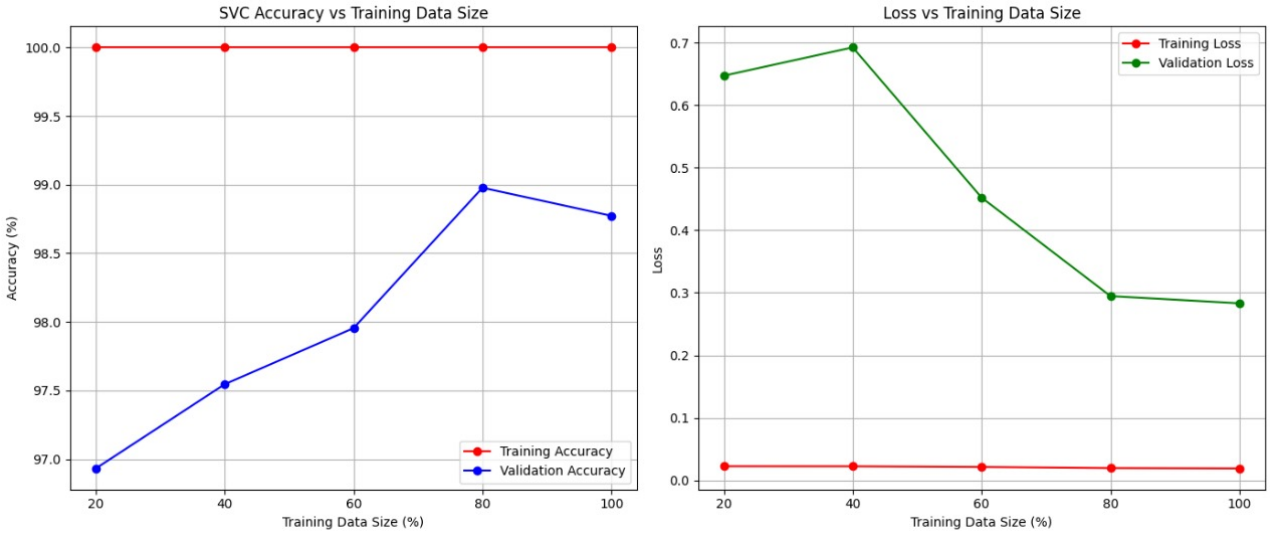
- **Model Development and Evaluation :**

- **Architecture :** A Support Vector Classifier (SVC) model was used for binary classification. The SVC uses the **Radial Basis Function (RBF)** kernel by default, which is effective for non-linear decision boundaries and works well with high-dimensional data.
- **Training :** The model was trained on the preprocessed training data, with the features scaled using **StandardScaler**. The SVC was trained using its default hyperparameters, including the RBF kernel. The training process involved fitting the model to the training set and optimizing the decision boundary to best separate the two classes.
- **Evaluation :** Once trained, the model's performance was evaluated on validation set. The primary evaluation metric used was **accuracy**, which represents the proportion of correct predictions made by the model on the validation data.

- **Accuracy vs Training Data Size :**

The model's validation accuracy on the validation dataset for different training dataset sizes is as follows:

- **Training size : 20% → Validation Accuracy : 96.93%**
- **Training size : 40% → Validation Accuracy : 97.55%**
- **Training size : 60% → Validation Accuracy : 97.96%**
- **Training size : 80% → Validation Accuracy : 98.98%**
- **Training size : 100% → Validation Accuracy : 98.77%**

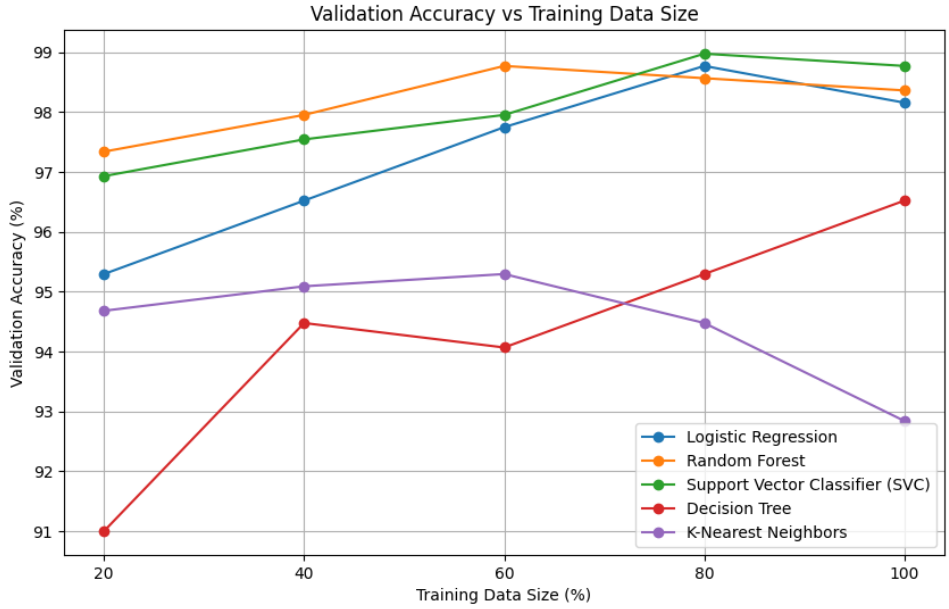


- **Other Models Tried :**

These models were also evaluated but were rejected due to lower accuracy compared to the final model.

- **Logistic Regression:** This model is simple yet effective for binary classification tasks like this one. It gives a decision boundary based on the features' linear relationships. It performed well, with a validation accuracy reaching up to 98.16%.
- **Random Forest:** By building multiple decision trees and averaging their predictions, this model captures complex interactions between the emoticon features. It improved with more data, achieving a peak accuracy of 98.36%.

- **Decision Tree:** This interpretable model works by splitting the data based on feature values, though it was more prone to overfitting, achieving a modest accuracy of 96.52%.
- **K-Nearest Neighbors (KNN):** KNN uses the majority class of the nearest neighbors to classify data points. It struggled with this high-dimensional dataset, resulting in the lowest accuracy of 92.84%.



Text Sequence Dataset :

This dataset consists of input features represented as strings of 50 digits. Each string captures a different encoding of the same input, and the goal is to classify each input into one of two classes (0 or 1).

- **Data Preprocessing :**

- **Converting Strings to Numerical Sequences :** The input_str column, which contains a string of digits, was converted into a numerical format. Each string was split into individual digits, and the digits were mapped into integers to form a sequence of numbers. This step was crucial for feeding the data into the neural network.

- **Model Development and Evaluation :**

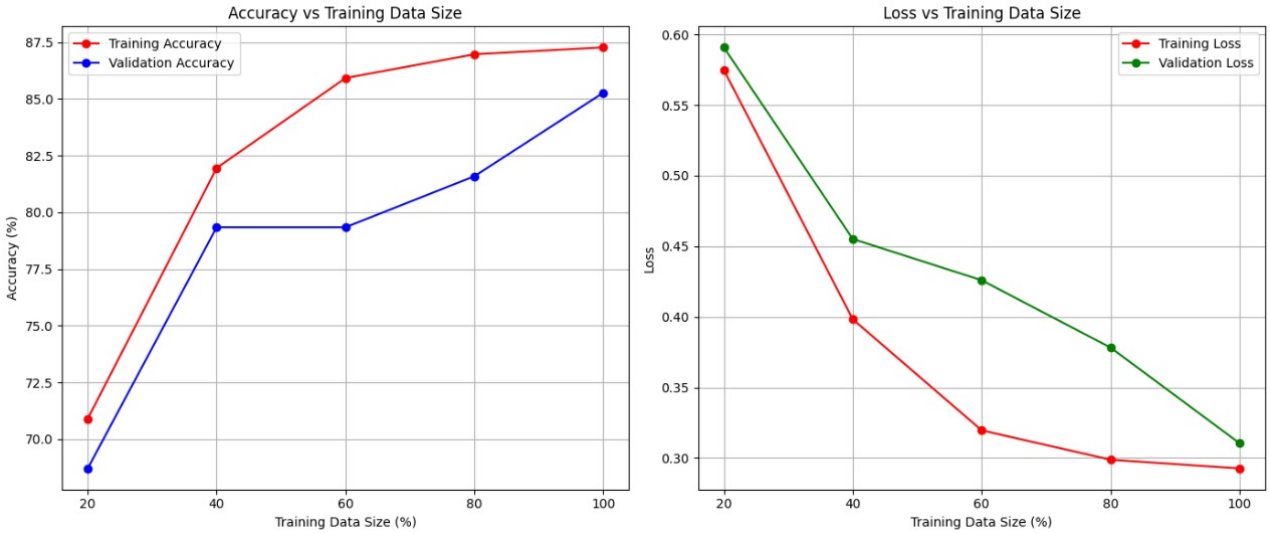
- **Architecture :** A custom Sequential model was built using the following layers:
  - ⋆ **Embedding Layer :** Converts input sequences into dense vectors of size 8.
  - ⋆ **Convolutional Layer :** A 1D convolutional layer with 32 filters and a kernel size of 5 to extract features from sequences.
  - ⋆ **GRU Layer :** A Gated Recurrent Unit (GRU) layer with 32 units to capture temporal dependencies in the sequence.
  - ⋆ **Dense Layers :** A fully connected layer with 16 units followed by an output layer with a sigmoid activation for binary classification.
- **Training :** The model was trained using the **Adam optimizer** with a learning rate of 0.001, a **binary cross-entropy** loss function, and accuracy as the evaluation metric. The training was conducted over 20 epochs with a batch size of 16.
- **Evaluation :** Once trained, the model's performance was evaluated on validation set. The primary evaluation metric used was **accuracy**, which represents the proportion of correct predictions made by the model on the validation data.
- **Trainable Parameters :** Total number of trainable parameters $\rightarrow$ **8,273**.

- **Accuracy vs Training Data Size :**

The model's validation accuracy on the validation dataset for different training dataset sizes is as follows:

- **Training size : 20% $\rightarrow$ Validation Accuracy : 68.71%**

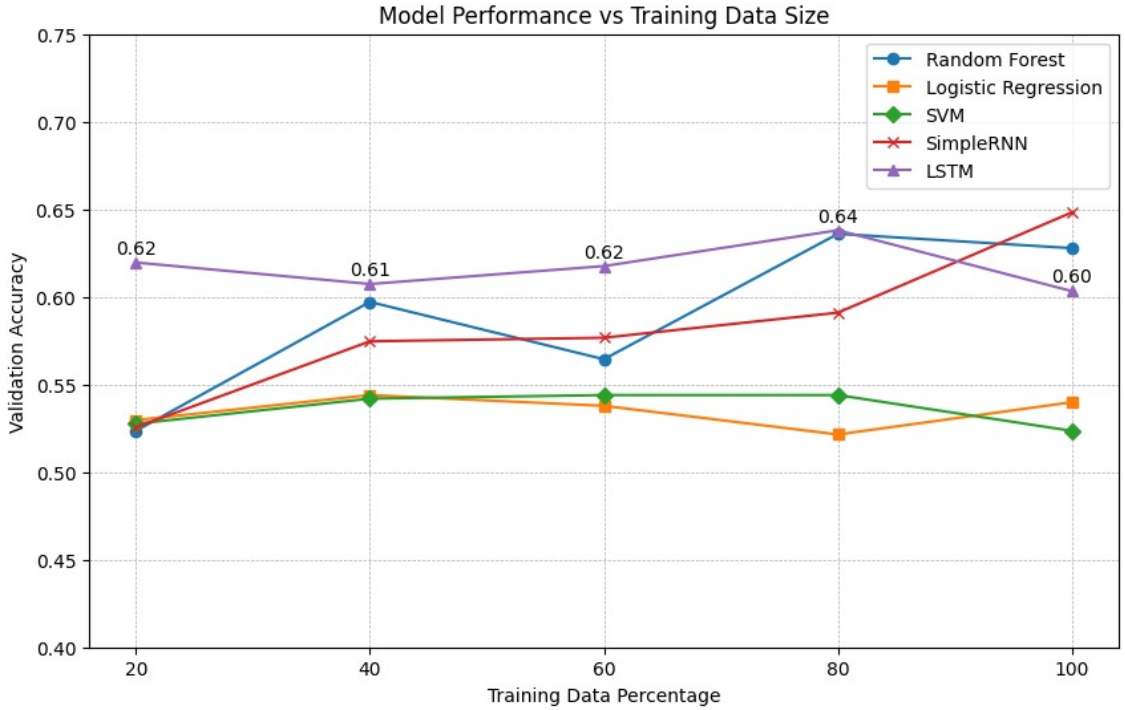
- **Training size : 40% → Validation Accuracy : 79.35%**
- **Training size : 60% → Validation Accuracy : 79.35%**
- **Training size : 80% → Validation Accuracy : 81.60%**
- **Training size : 100% → Validation Accuracy : 85.28%**



- Other Models Tried :

These models were also evaluated but were rejected due to lower accuracy compared to the final model.

- **Random Forest :** Achieved 63% accuracy. It performed well with non-sequential data but struggled with patterns in sequential data.
- **Logisitic Regression :** Reached an accuracy of 54%. While effective in binary classification, it couldn't model sequence dependencies.
- **Support Vector Machine :** Attained an accuracy of 52.5%. SVM's computational complexity was an issue as dataset size increased.
- **SimpleRNN :** Achieved 65% accuracy. It captured some sequential patterns but was prone to vanishing gradients.
- **LSTM :** Initially, the LSTM reached 60% accuracy by capturing more long-term dependencies than SimpleRNN.



Task 2

In this task, the goal is to investigate whether combining the three different feature representations of the same dataset leads to a more accurate model compared to using each dataset individually. The three training datasets contain the same inputs, but they differ in how the features are represented. By merging these distinct representations, we aim to capture complementary information from each dataset, potentially resulting in a more robust and accurate model.

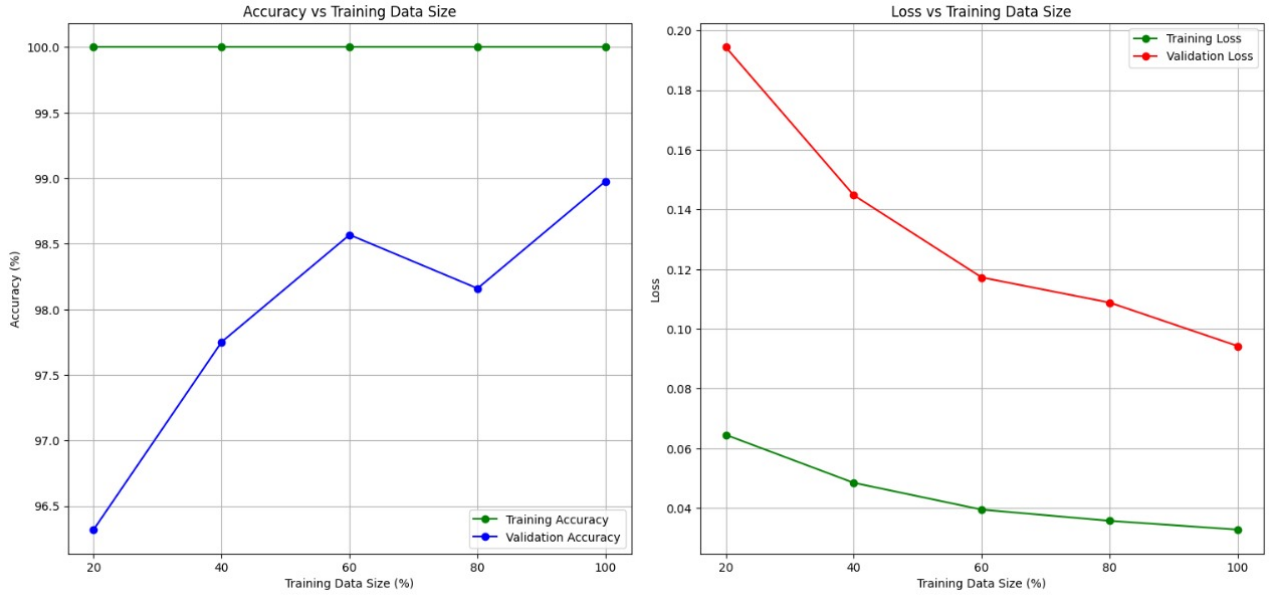
- **Data Preprocessing :**

  - **Emoticons Dataset :** The categorical emoticon features are one-hot encoded using **OneHotEncoder**. This converts the 13 categorical features into binary vectors.
  - **Deep Features Dataset :**
    - **Flattening :** The deep features matrix (13x768) is flattened to a 1D array of size 768 for each input using .reshape(). This ensures all the feature vectors have the same length.
    - **Scaling :** The flattened features are scaled using StandardScaler to normalize the data, ensuring each feature has zero mean and unit variance.
  - **Text Sequence Dataset :**
    - **Tokenization :** The 50-character strings are tokenized using Keras' Tokenizer with char_level=True. This converts the string of characters into a matrix of binary values, where each character in the string is represented by a binary vector.
    - **One-hot Encoding :** The tokenizer converts the sequences into one-hot encoded format using texts_to_matrix with mode='binary', resulting in a binary matrix where each character is represented by a binary vector.
  - **Combining Features :** After preprocessing each dataset, the features are concatenated (i.e., combined) along the last axis using np.concatenate(). This creates a unified feature matrix where each input has a combined representation from emoticons, deep features, and text sequences.
  - **Scaling Combined Data :** The combined features are then scaled using StandardScaler to ensure uniform scaling across all datasets. This helps the model converge faster and perform better.

- **Accuracy vs Training Data Size :**

The model's validation accuracy on the validation dataset for different training dataset sizes is as follows:

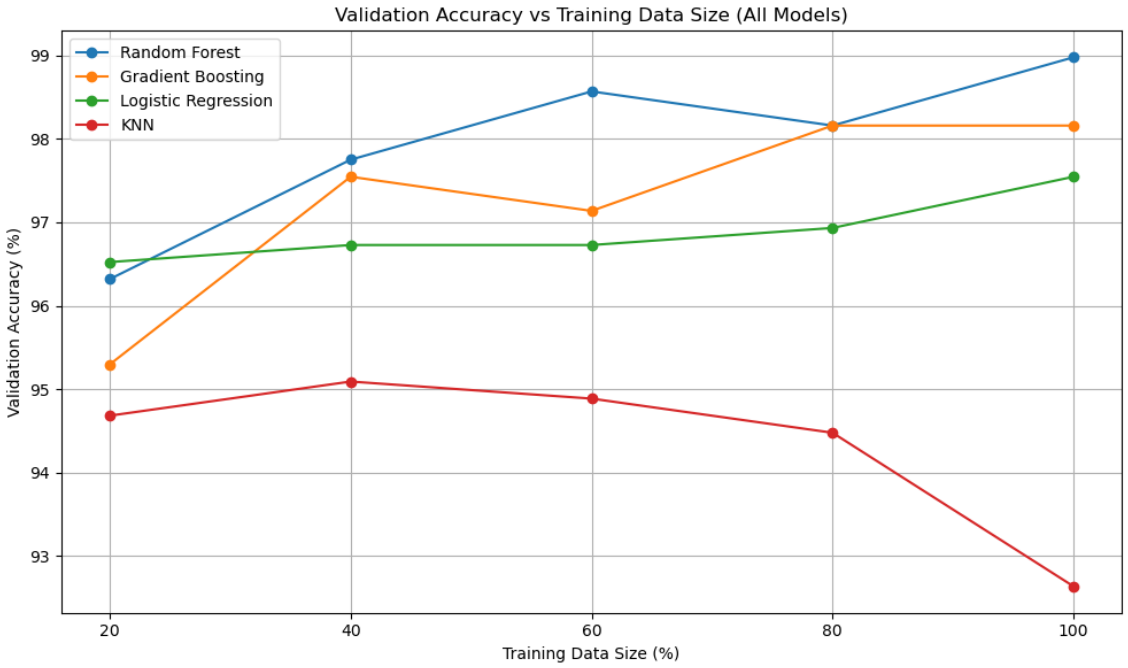
- **Training size : 20% → Validation Accuracy : 96.32%**
- **Training size : 40% → Validation Accuracy : 97.75%**
- **Training size : 60% → Validation Accuracy : 98.57%**
- **Training size : 80% → Validation Accuracy : 98.16%**
- **Training size : 100% → Validation Accuracy : 98.98%**



- **Other Models Tried :**

These models were also evaluated but were rejected due to lower accuracy compared to the final model.

- **Logistic Regression:** This model is straightforward and effective for binary classification tasks. It uses a linear decision boundary based on the feature relationships. In this evaluation, it achieved a maximum validation accuracy of 97.55%, demonstrating decent performance but not quite matching the top models.
- **Gradient Boosting:** This model builds trees sequentially to correct errors made by previous models. It provided stable performance, with a maximum validation accuracy of 98.16%. However, it did not outperform the Random Forest model, making it less favorable for this task.
- **K-Nearest Neighbors (KNN):** KNN classifies data points based on the majority class of their nearest neighbors. While intuitive, it struggled with the high dimensionality of the combined dataset, resulting in the lowest accuracy of 92.64%. This model's performance was affected by the feature space complexity.



References :

- **SimpleRNN layer :** https://keras.io/api/layers/recurrent_layers/simple_rnn/
- **Understanding LSTM Networks :** https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- **StandardScaler :** https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- **Conv1D layer :** https://keras.io/api/layers/convolution_layers/convolution1d/
- **Convolutional Neural Networks :** https://kitchell.github.io/DeepLearningTutorial/4cnnsinkeras.html
- **Understanding Gated Recurrent Unit (GRU) in Deep Learning :** https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2