Mini-Project 1

Name of Group: **Error 404**

Group Members:

Ahmad Raza	220088
Harsh Kumar	220428
Kanav Singh Chouhan	220495
Sankalp Pande	220964

Indian Institute of Technology Kanpur

CS771: Introduction to Machine Learning

Instructor: Prof. Piyush Rai

October 22, 2024

1 Task1: Binary Classification Models

We aim to train binary classification models on three datasets representing the same task but with varying feature representations. The goal is to select the best-performing model for each dataset in terms of accuracy and the amount of training data required. We trained and evaluated three models: Logistic Regression, Decision Tree, and Stochastic Gradient Descent, and compared their performance using validation data.

1.1 Emoticons as Features Dataset

1.1.1 Introduction

Here, we aimed to develop and evaluate binary classification models for a dataset where the features were emoticons extracted from text. The objective was to predict a label based on these features. We implemented and evaluated **Logistic Regression** and the following two versions of **Stochastic Gradient Descent (SGD)** model:

- **SGD without Feature Selection**: In this version, all emojis were treated as features and used for training the model.
- **SGD with Feature Selection**: Here, we removed the top 7 most frequent emojis to reduce noise and potentially improve model performance.

1.1.2 Datasets and Preprocessing

The dataset consists of a column input_emoticon, which contains a string of emojis, and a label column that represents the class. We preprocessed the data in the following ways:

1. **Logistic Regression**:
   - Splitting Emoticons: The input_emoticon strings were split into individual columns for each emoji, facilitating separate feature extraction. This was done for both training and validation datasets.
   - Combining Data: The split emoticon columns were combined with their corresponding labels to create final dataframes for training and validation.
2. **SGD without Feature Selection**: We split each row of the input_emoticon column into individual emojis and treated them as features using one-hot encoding. This approach kept all the emojis as features.
3. **SGD with Feature Selection**: We analyzed the frequency of each emoji in the dataset and identified the top 7 most frequent emojis. We then removed these frequent emojis from each input, reasoning that their presence might not carry enough distinctive information for classification.

1.1.3 Models and Methods

- **One-Hot Encoding**: In all the three models, we used the OneHotEncoder to transform the categorical emoji features into numerical vectors suitable for training.
- **GridSearchCV**: We performed hyperparameter tuning using GridSearchCV, testing various combinations of:
  1. Loss functions (modified_huber, perceptron, hinge, log_loss),
  2. Penalty types (l2, l1, elasticnet),
  3. Regularization strengths (alpha values: 0.01, 0.1, 1),
  4. Learning rates (constant, optimal, invscaling).

After tuning, the best set of hyperparameters was selected for further training and evaluation.

1.1.4 Experimentation

To evaluate the models, we experimented with different fractions of the training data to analyze how the amount of training data affects the model's accuracy.

1. Training Data Fractions: We trained the models on 20%, 40%, ..., up to 100% of the available data.
2. Metrics: We recorded both training accuracy and validation accuracy at each fraction of the data to assess overfitting or underfitting.
3. Parameter Count: We also calculated the number of parameters (features + intercepts) used by the model, to measure the complexity of the trained models.

1.1.5 Results

	Validation Accuracy		
Training Size(%)	LogisticRegression	SGD(No FeatureSelection)	SGD(With FeatureSelection)
20	0.6994	0.6789	0.8180
40	0.8221	0.7546	0.9018
60	0.8834	0.8057	0.9202
80	0.9202	0.8037	0.9550
100	0.9284	0.8650	0.9652

Table 1: Validation Accuracies for Logistic Regression and SGD

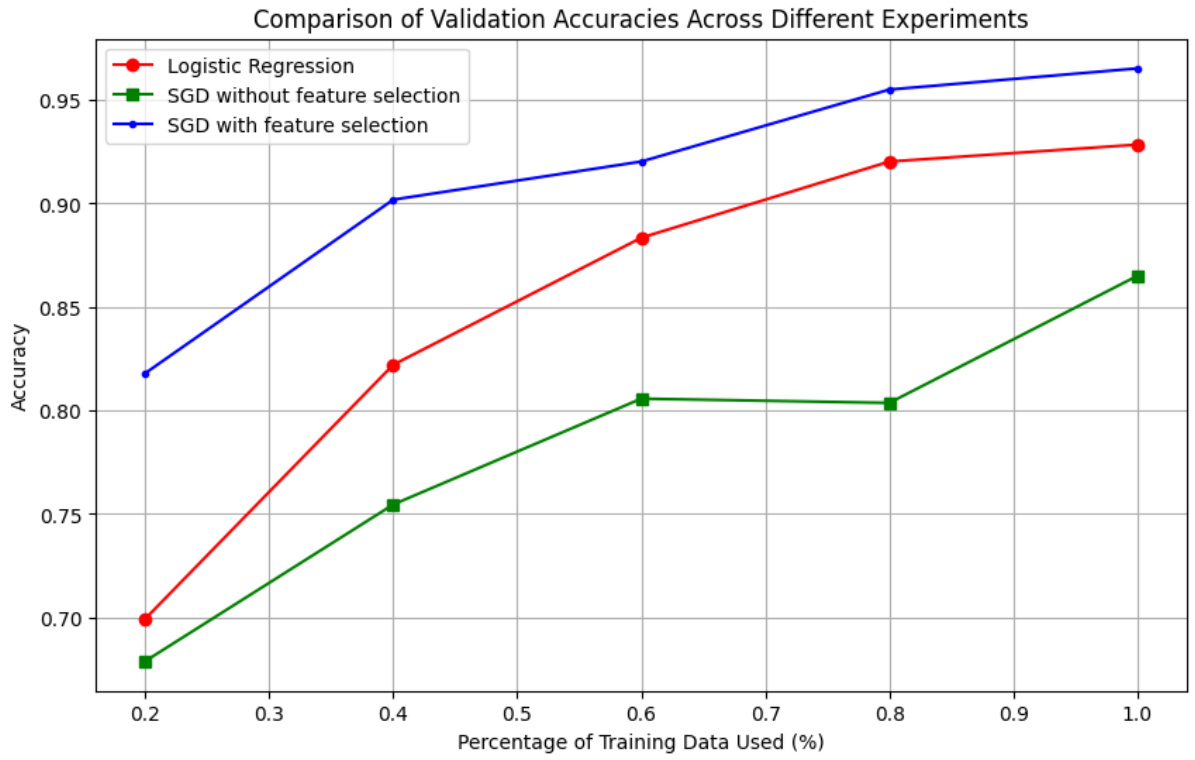

Figure 1: Validation Accuracy vs. Training Set Size for Different Models

1.1.6 Conclusion

Best model: SGD model with feature selection. This model is giving around 96% accuracy on validation set for 100% training data.

1.2 Deep Features Dataset

1.2.1 Introduction

This task focuses on training and evaluating multiple classification models on a dataset represented by deep features. These features are embeddings derived from a pre-trained model. The goal is to compare the performance of different machine learning models on this dataset and understand how the size of the training set impacts model performance.

1.2.2 Data Preprocessing

- **Data Structure**: The features in this dataset are represented as embeddings of shape (samples, 13, 786), which were flattened into a 1-dimensional vector per sample.
- **Normalization**: After flattening, the features were standardized using StandardScaler to ensure all features had a mean of 0 and a variance of 1. This step helps the models converge faster and perform better, especially for models like SVM and Logistic Regression.

1.2.3 Models

1. **Logistic Regression**: A linear model, useful as a baseline for classification tasks. We used the lbfgs solver and set max_iter=1000 to ensure convergence.
2. **K-Nearest Neighbors (KNN)**: A distance-based model where we used GridSearchCV to tune hyperparameters such as n_neighbors and weights.
3. **Support Vector Machine (SVM)**: A linear kernel SVM was trained to evaluate its performance on the flattened embeddings.
4. **Random Forest**: A powerful ensemble learning model with regularization to control overfitting. We used parameters like n_estimators=100, max_depth=10, min_samples_split=5, and min_samples_leaf=2.

1.2.4 Experimentation

We experimented with training each model on different fractions of the training data, ranging from 20% to 100%, to observe how the training size impacts model performance. The following metrics were recorded for each model:

- Accuracy: Proportion of correct predictions on the validation data.
- Confusion Matrix: A matrix showing the distribution of predicted vs. actual labels.
- Classification Report: Includes precision, recall, F1-score, and support for each class.

Hyperparameter Tuning:

- KNN: We applied GridSearchCV for hyperparameter tuning, specifically targeting the n_neighbors and weights parameters to optimize the model's performance.

1.2.5 Results

	Validation Accuracy			
Training Size(%)	LogisticRegression	KNN, Weighted & Tuned	SVCClassifier	RandomForest
20	0.9530	0.9530	0.9448	0.96
40	0.9652	0.9673	0.9673	0.97
60	0.9775	0.9550	0.9714	0.98
80	0.9877	0.9632	0.9796	0.98
100	0.9816	0.9591	0.9796	0.98

Table 2: Validation Accuracies for Logistic Regression, KNN, SVC and SGD Classifiers

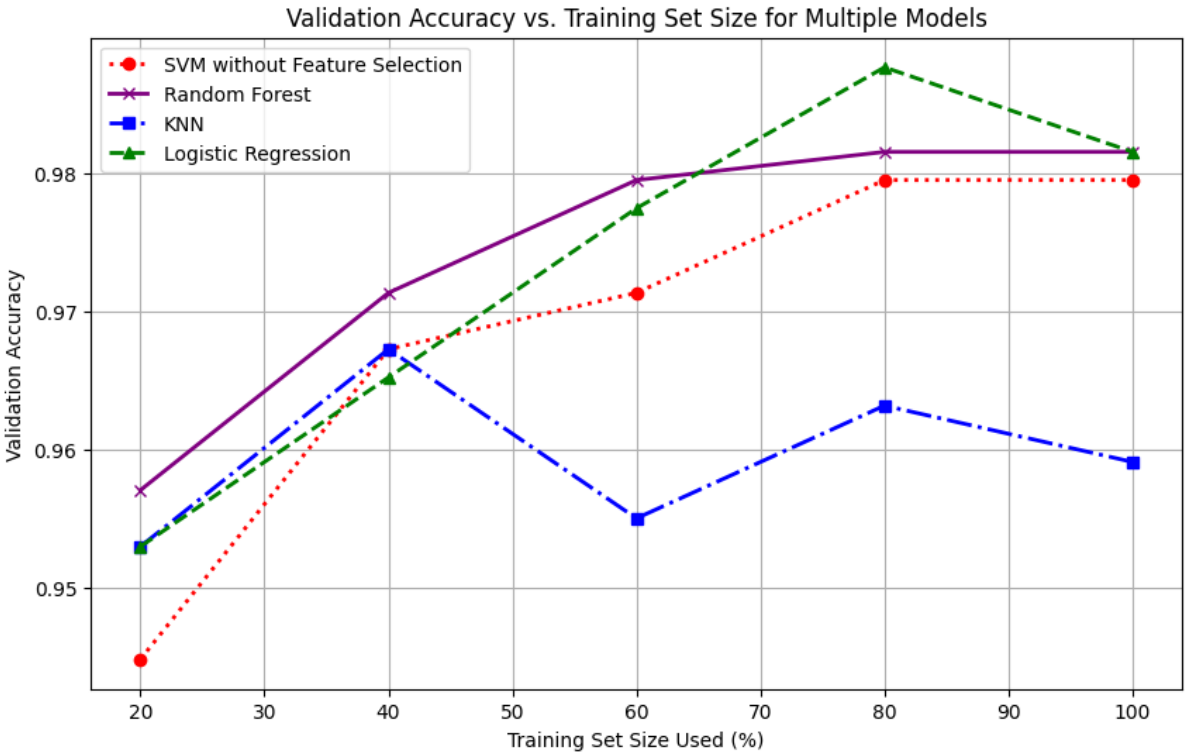

Figure 2: Validation Accuracy vs. Training Set Size for Different Models

1.2.6 Conclusion

Random Forest is taken as the ideal model performing better at maximum instances.

1.3 Text sequence Dataset

1.3.1 Introduction

Here, we aimed to explore the effectiveness of different machine learning models in classifying sequences of characters. The primary focus was on comparing the performance of XGBoost with and without hyperparameters against a Long Short-Term Memory (LSTM) neural network. The goal was to determine which model achieved the highest accuracy in binary classification while examining the impact of training data fractions on performance.

1.3.2 Dataset Description

The datasets used for training and validation were sourced from CSV files containing sequences of characters labeled for binary classification. Preprocessing steps included:

1. Removing Leading Zeros: Sequences were stripped of leading characters.
2. Filtering Substrings: Specific substrings deemed irrelevant were removed from the sequences.
3. String Length Validation: Only sequences with a modified length of 13 were retained for further analysis.

The resulting datasets were then encoded, converting characters into integer representations for input into the models.

1.3.3 Models

1. **XGBoost (Without Hyperparameters)**: The XGBoost model was initially implemented using default hyperparameters. The training process involved fitting the model to the training dataset and evaluating it on the validation set. The key parameters included:
   - Learning Rate: Default value (0.3)
   - Maximum Depth: Default value (6)
2. **XGBoost (With Hyperparameters)**: To enhance the performance of the XGBoost model, hyperparameters were tuned using RandomizedSearchCV. This involved searching through various combinations of parameters such as:
   - Learning Rate
   - Maximum Depth
   - Number of Estimators
   - Subsample Ratio

   The goal was to minimize overfitting and improve generalization on the validation set.
3. **LSTM**: The LSTM model was configured with the following architecture:
   - Embedding Layer: Transformed integer-encoded input sequences into dense vectors of fixed size (32).
   - LSTM Layer: A single LSTM layer with 32 units to capture temporal dependencies.
   - Dense Layers: Two dense layers with 16 and 8 units, respectively, using ReLU activation.
   - Output Layer: A final dense layer with a sigmoid activation function for binary classification.

   The model was compiled using the Adam optimizer and binary crossentropy loss function. The training was conducted over 50 epochs.

1.3.4 Training and Evaluation

The training process for each model involved evaluating their performance based on varying fractions of the training data. The data fractions included: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]. Training accuracy and Validation accuracy are recorded.

1.3.5 Results

		Validation Accuracy	
Training Size(%)	XGBoost (Without Hyperparameters)	XGBoost (With Hyperparameters)	LSTM
20	0.7918	0.7701	0.7310
40	0.8395	0.7983	0.8069
60	0.8525	0.8178	0.8134
80	0.8764	0.8243	0.8503
100	0.8720	0.8286	0.8764

Table 3: Validation Accuracies for XGBoost and LSTM Classifiers

1.3.6 Conclusion

Model giving best results among tested ones -: XgBoosting (Without Hyperparameters)
Inference: Xgboosting with hyperparameter doesn't perform better than without H.parameter since it causes model to undergo overfitting thus reduce performance on testing.

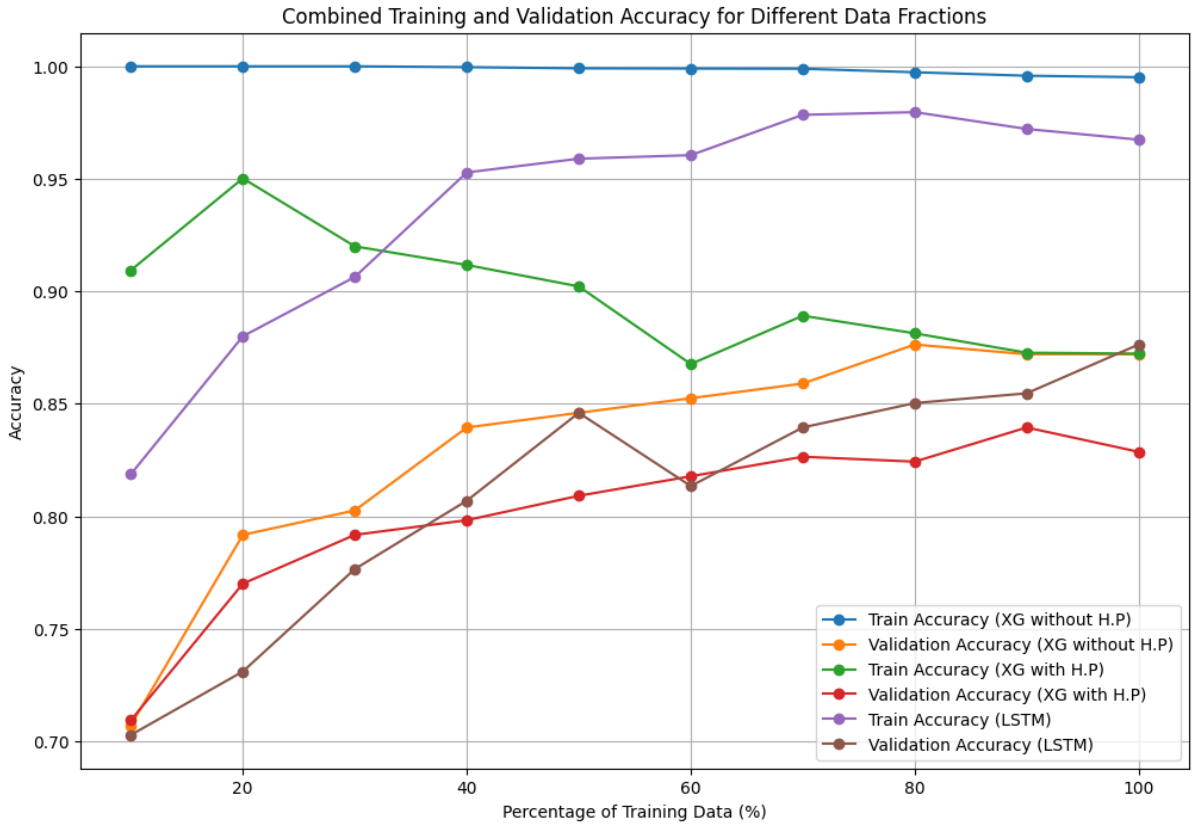

Figure 3: Validation and Training Accuracy vs. Training Set Size for Different Models

2 Task 2: Combining Datasets for Improved Performance

2.1 Approach to Combining Datasets:

2.1.1 (a). Combining Features Across Datasets

Each dataset represents different feature sets for the same input. The strategy was to merge these feature representations to provide a more comprehensive view of each instance:

- **Emoticon Dataset (13 Features)**: The Emoticon dataset contains categorical features. These were transformed into numerical format using encoding techniques like one-hot encoding or label encoding.
- **Text Sequence Dataset (50 Features)**: The Text Sequence dataset represents inputs as a sequence of 50-length strings. Feature extraction was applied to convert these into numerical representations, including techniques such as character-level embeddings or statistical features like character frequency.
- **Deep Features Dataset (768 Features)**: This dataset already consists of high-dimensional numerical embeddings, generated by a neural network. These features were directly used in combination with the other datasets.
- **Feature Matrices Shape**: Feature matrix shape for combined **Training Dataset (7080, 13, 768)** Feature matrix shape for combined **Validation Dataset (489, 13, 768)**

2.1.2 (b). Merging Datasets

Once each dataset's features were processed, we concatenated them horizontally for each corresponding input. This involved combining the 13 features from the Emoticon dataset, 50 features from the Text Sequence dataset, and 786 features from the Deep Features dataset into a single input vector for each instance, resulting in a combined feature vector of size $13 + 50 + 768 = \mathbf{831}$ features.

Ensuring that the rows of the datasets were aligned properly was critical, so that features from different representations corresponded to the same input.

2.1.3 (c). Model Selection for Combined Dataset

After combining the datasets, we trained models on this unified feature set. The models explored include:

- **Logistic Regression**: Suitable for binary classification, especially with large datasets.
- **Neural Networks**: Given the size and complexity of the combined dataset, neural networks could also be an appropriate choice.
- **Random Forests**: Capable of handling large feature sets and extracting useful patterns from diverse feature spaces.

2.2 Training and Results on Combined Dataset

To optimize model performance and assess the impact of training data size on accuracy, we experimented with a **Random Forest classifier** using varying fractions of the combined dataset. This process aimed to determine the best balance between model complexity and generalization, particularly by controlling overfitting through regularization.

2.2.1 Model Configuration

The Random Forest model was configured with the following hyperparameters to enforce regularization:

- **Number of estimators**: 100 trees were used in the forest to balance performance and computational efficiency.
- **Maximum tree depth**: Set to 8, limiting the complexity of each decision tree to prevent overfitting.
- **Minimum samples split**: Increased to 10, requiring more samples to split a node, which helps control overfitting.

- **Minimum samples per leaf**: Set to 5, ensuring that each leaf in the tree has a minimum of 5 samples, adding regularization.
- **Maximum features**: Set to 'sqrt', limiting the number of features considered at each split to reduce model variance.
- **Bootstrap sampling**: Disabled, allowing the model to use all available data for each tree to increase stability.

2.2.2 Experimental Process

We trained the model on subsets of the training data, using fractions ranging from 20% to 100% of the total data. For each fraction, the model was trained on the selected subset and evaluated on both the subset and the full validation set.

1. **Training subset evaluation**: The model's accuracy was measured on the subset it was trained on to assess how well it learned from the reduced data.
2. **Validation set evaluation**: The model was then evaluated on the full validation set to assess how well it generalizes to unseen data.

2.2.3 Results

Model	Training Set Size(%)	Total Features(Combined)	Training Accuracy	Validation Accuracy
RandomForest(Reg)	20	831	1.0000	0.9591
RandomForest(Reg)	40	831	0.9996	0.9714
RandomForest(Reg)	60	831	0.9991	0.9775
RandomForest(Reg)	80	831	0.9993	0.9836
RandomForest(Reg)	100	831	0.9993	0.9836

Table 4: Validation Accuracies of Regularized Random Forest

- **Comparison with Individual Models**: Compare the performance of the combined model with the individual models from Task 1(fig. 1).

2.2.4 Conclusion

Best model: Regularized Random Forest performed best with large combined dataset with 98.4% accuracy

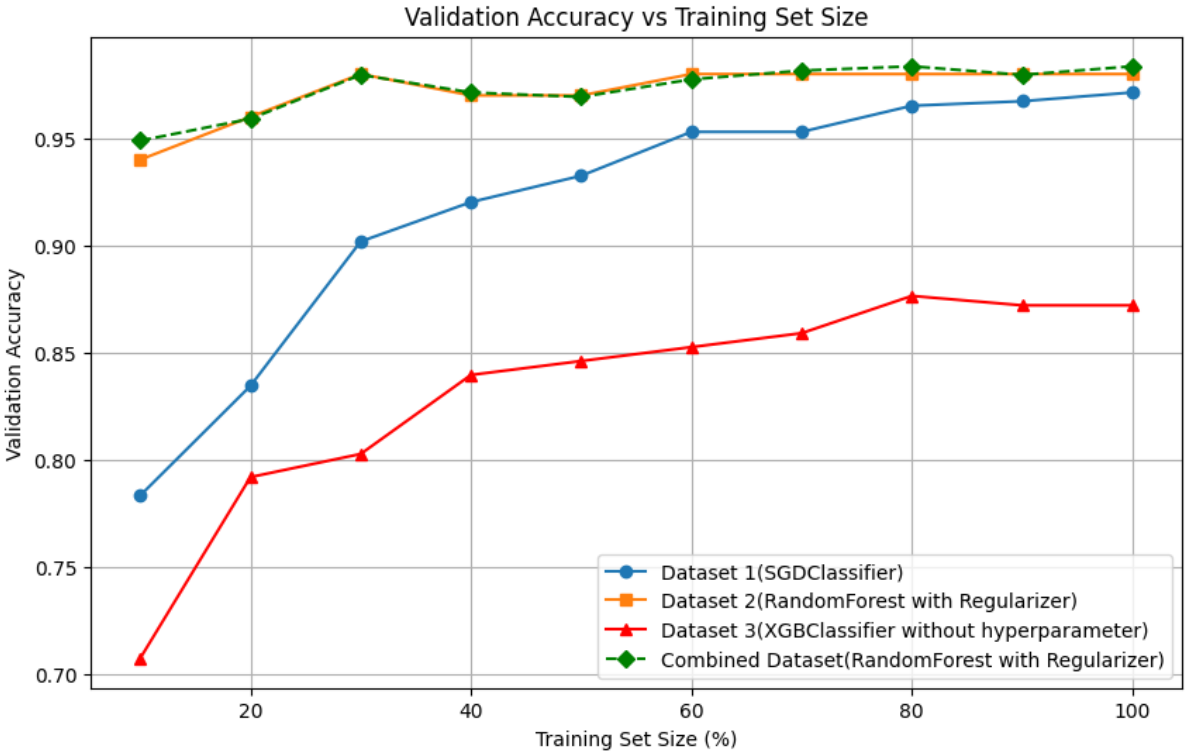

Figure 4: Validation Accuracy vs. Training Set Size for Different Models