# Mini-Project 1 CS771

**Purav Jangir**
220837

**Prerak Agarwal**
220818

**Burhanuddin Merchant**
220300

**Priyanshu Maurya**
220827

**Lovedeep Sharma**
220595

## 1 Abstract

In this project, we explored different feature representations of a dataset derived from the same parent data to understand their impact on predictive modeling. The process began with thorough **preprocessing** and analysis of three distinct datasets, each possessing unique feature characteristics. Utilizing these datasets, we implemented a variety of machine learning models, including Logistic Regression, Support Vector Machine, Random Forest, `LightGBM`[1], and `Long Short-Term Memory networks`[2]. Extensive hyperparameter tuning was conducted for each model to optimize performance. We focused on visualizing accuracy trends throughout the experimentation to manage the trade-off between bias and variance. Additionally, a combined model was developed by integrating the features from all three datasets, leveraging the strengths of each representation. The final analysis compared the performance of these approaches, demonstrating the effectiveness of combining multiple feature sets to improve model accuracy.

## 2 Datasets

The project utilized **three** distinct datasets derived from the same parent dataset, each with unique feature representations:

### 2.1 Emoticon Dataset

This dataset consists of 13 emoticons representing each input. These emoticons encode essential features derived from the raw data, providing a compact symbolic representation.

#### 2.1.1 Dataset Preprocessing

- `Character Frequency Analysis`: Character counts were conducted across the dataset, resulting in the identification of the top seven most frequent characters. These characters were subsequently removed to mitigate potential bias, as their presence was consistent in every input. While this approach adheres to one-hot encoding logic and aims to reduce bias, it is important to note that it overlooks the significance of the order among the emojis.
- `Unicode Mapping`: A dictionary mapping each emoticon character to its Unicode index (from U+1F600 to U+1F64F) was created. Any character outside this range was mapped to -1. This range is analyzed from the dataset itself.
- `Truncation and Padding`: Emoticons were truncated or padded to a length of 3 Unicode values. Emoticons shorter than 3 characters were padded with -1.

[1] `https://lightgbm.readthedocs.io/`
[2] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM`

- `Feature Creation`: The three encoded Unicode values were stored in columns **`encoded_1`**, **`encoded_2`**, and **`encoded_3`**. These processed columns were used as features, and the target column was the binary label.

### 2.1.2 Analysis on Different Models Tried

We initially experimented with various models, including Binary Classification, Support Vector Machine, and Random Forest. However, we ultimately selected **LightGBM** due to its superior performance in handling sparse data, and efficient gradient boosting capabilities, which collectively contribute to better accuracy and scalability for our dataset.

| **Model** | **Accuracy (%)** |
|---|---|
| Logistic Regression | 57.06 |
| SVM | 58.69 |
| Random Forest | 72.80 |
| LightGBM | 95.91 |

Table 1: Model Accuracies

### 2.1.3 Performance Analysis on LightGBM

1. `Hyper-Parameters`:
   - `objective`: binary,
   - `metric`: binary_logloss,
   - `boosting type`: gbdt,
   - `learning_rate`: 0.45,
   - `num_leaves`: 31,
   - `feature_fraction`: 0.3.

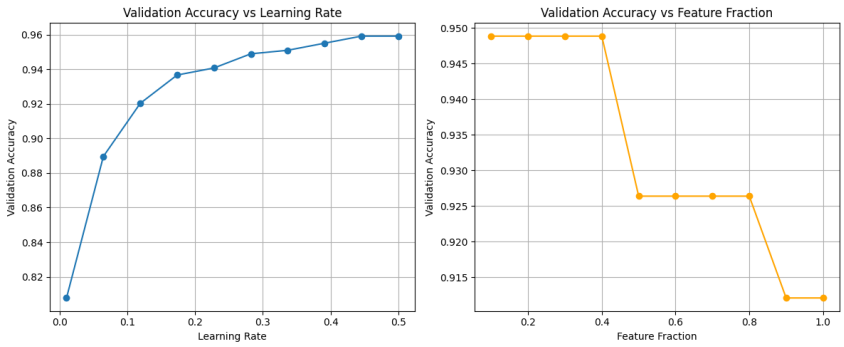



Figure 1: Validation Accuracy vs Learning Rate (*on left*) and Validation Accuracy vs Feature Fraction (*on right*)

2. `Training`: Trained with 105 boosting iterations on the training set, using the validation set for evaluation.

3. `Training Data Usage vs. Validation Accuracy`: As shown in the provided plot, validation accuracy increased with the percentage of training data used. Starting from 74.23% with 10% of data, the model reached an early peak of 95.09% when trained with **70%** of the data. Slight variations in performance were seen beyond 80% of data usage, stabilizing around 94.68% to 95.91%.

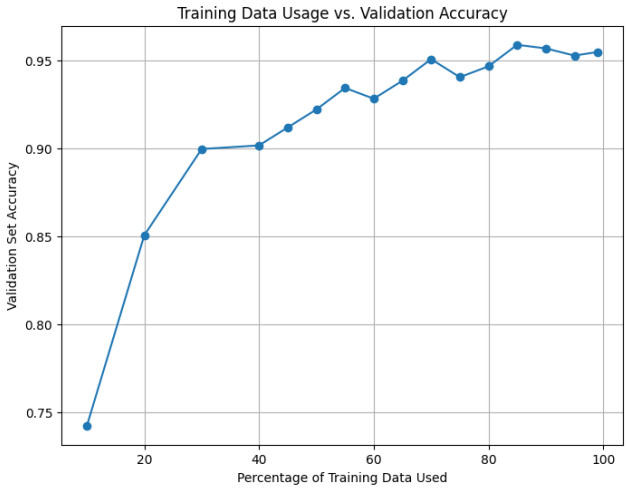



Figure 2: Plot for Training Data Usage vs Validation Accuracy

#### 2.1.4 Final Results

- `Validation Accuracy`: 95.91%.
- `Precision and Recall`: Both classes achieved around 96.8%, indicating balanced performance across categories.
- `Confusion Matrix`:
  - Class 0: 244 correct predictions, 8 misclassified.
  - Class 1: 225 correct predictions, 12 misclassified.
- `F1-Score`: The F1-scores for both classes were 0.96, indicating a well-balanced model with minimal bias-variance tradeoff.

### 2.2 Deep-Features Dataset

Each input is represented by a 13x786 matrix, capturing richer and more granular details through high-dimensional embeddings. This representation aims to retain detailed information for deep feature extraction.

#### 2.2.1 Dataset Preprocessing

- `Concatenation of Features`: The 13x786 matrices were flattened to form a single 10,218-dimensional vector per input, combining all feature embeddings into a single input vector.
- `Scaling`: A StandardScaler was applied to normalize the input data, ensuring that all features contributed equally to model training.
- `Dimensionality Reduction`: To reduce computational complexity and focus on the most relevant features, **Principal Component Analysis** was performed, reducing the dimensionality of the input from 10,218 to 300. As the CEV ratio approaches 1 near 300, it indicates that the necessary information is effectively captured within this number of components.

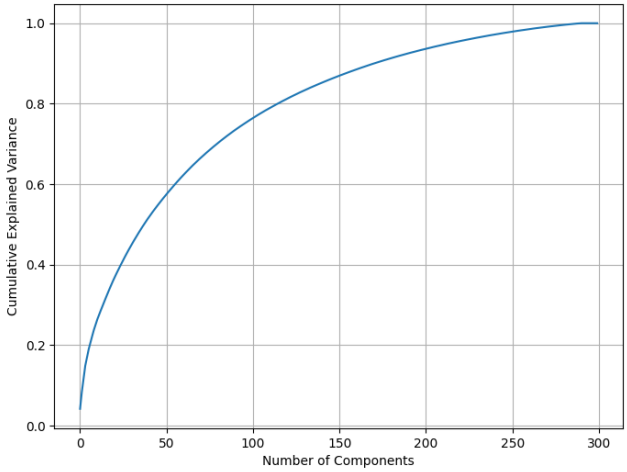



Figure 3: Cumulative Explained Variance vs Number of Components

### 2.2.2 Analysis on Different Models Tried

For this dataset, we also tested various models, with their accuracies presented in Table 2. We chose **LightGBM** due to its efficient gradient-boosting capabilities.

| **Model** | **Accuracy (%)** |
|---|---|
| Logistic Regression | 97.16 |
| SVM | 97.17 |
| Random Forest | 96.32 |
| LightGBM | 97.55 |

Table 2: Model Accuracies

### 2.2.3 Performance Analysis on LightGBM

1. `Hyper-Parameters`:
   - `objective`: binary,
   - `metric`: binary_logloss,
   - `boosting type`: gbdt,
   - `learning_rate`: 0.2,
   - `num_leaves`: 31,
   - `feature_fraction`: 0.4.

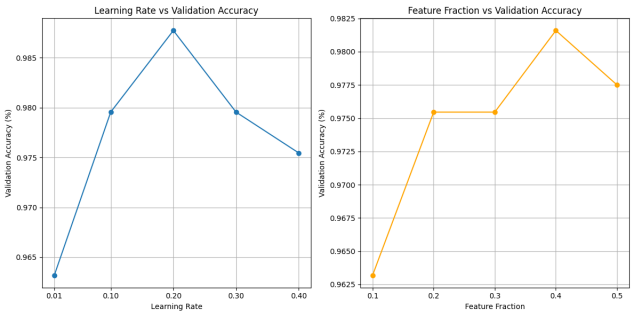



Figure 4: Validation Accuracy vs Learning Rate (*on left*) and Validation Accuracy vs Feature Fraction (*on right*)

2. `Training Data Usage vs. Validation Accuracy`: The graph shows validation accuracy improving as more training data is used. Starting around 93.2% with 10% of the data, accuracy rises sharply to about 97% at 30%. After some fluctuations, it peaks at **98.16% with 70%** of the data, stabilizing between 97.5% and 98% beyond that. Overall, more data improves accuracy, but the gains diminish after 80%.

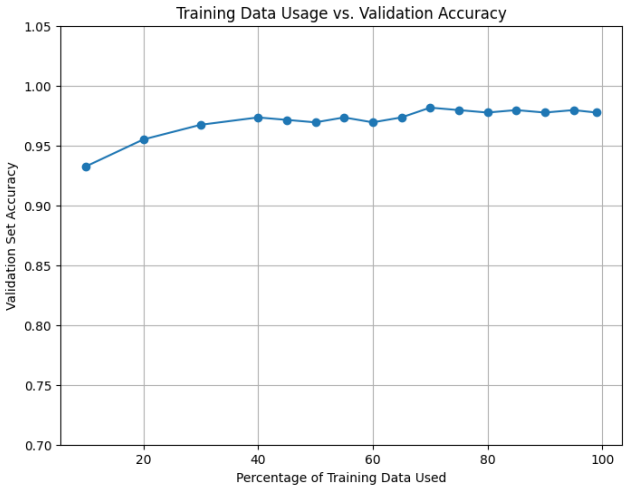



Figure 5: Plot for Training Data Usage vs Validation Accuracy

#### 2.2.4 Final Results

- `Validation Accuracy`: 97.55%.
- `Precision and Recall`: Both classes achieved around 0.97 and 0.98, indicating balanced performance across categories.
- `Confusion Matrix`:
  - Class 0: 245 correct predictions, 7 misclassified.
  - Class 1: 232 correct predictions, 5 misclassified.
- `F1-Score`: The F1-scores for both classes were 0.98.

### 2.3 Text Sequence Dataset

This dataset contains strings of fixed length 50, where each string is made up of digits ranging from 0 to 9. The goal is to predict a binary label indicating the presence or absence of a particular characteristic in the sequence.

#### 2.3.1 Dataset Preprocessing

- `Initial Observations`: The first three digits of each string in the dataset were zeros, so we trimmed the first 3 characters of each sequence to remove redundant information.
- `Character Tokenization`: A **Tokenizer** from the *tensorflow* library was applied to convert the character-level input strings into numerical tokens. These tokenized sequences were stored in the `X_train_seq` dataset.
- `Padding and Truncation`:To ensure uniformity, all sequences were padded or truncated to have a consistent length. This step was done using the `pad_sequences` function from **Keras**, resulting in the final training data, `X_train_textseq`.

#### 2.3.2 Analysis on Different Models Tried

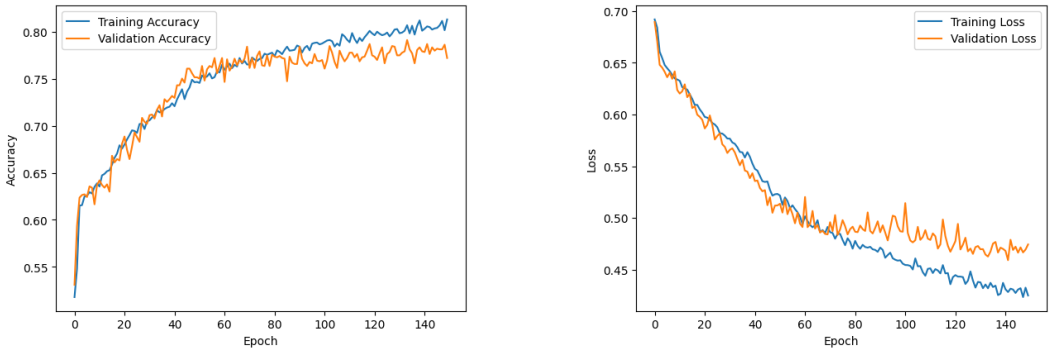
We evaluated various models for the `text_seq` dataset. After comparing the performance, we chose to proceed with the **LSTM model**.

| Model | Accuracy (%) |
|---|---|
| Logistic Regression | 52.47 |
| SVM | 55.32 |
| Random Forest | 63.57 |
| LightGBM | 69.94 |
| Long Short-Term Memory (LSTM) | 76.89 |

Table 3: Model Accuracies

### 2.3.3 Performance Analysis on LSTM Model

1. `Hyper-Parameters`:
   - `Output dimension`: 64
   - `LSTM units`: 8
   - `Dropout`: 20%
   - `Dense layer units`: 1
   - `Activation function`: Sigmoid
   - `Epoches`: 150




(a) Validation Acc. and Training Acc. vs. Epochs

(b) Validation Loss and Training Loss vs. Epochs

Figure 6: Comparison of Validation and Training metrics: Accuracy (*left*) and Loss (*right*)

2. `Training Data Usage vs. Validation Accuracy`: As training progressed, validation accuracy increased with the percentage of data used, peaking at 78.32% when trained on 100% of the dataset.

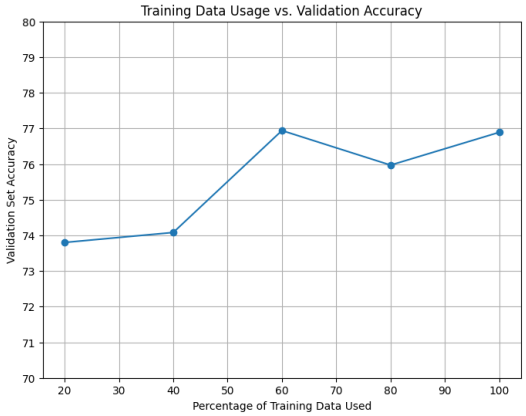



Figure 7: Plot for Training Data Usage vs Validation Accuracy

#### 2.3.4 Final Results

- `Validation Accuracy`: 76.89%.
- `Performance Observation`: The LSTM model consistently outperformed other models like LightGBM and Support Vector Machine, highlighting its effectiveness in capturing long-term dependencies within the `text_seq` dataset.

### 2.4 Combined Dataset

Each dataset underwent individual pre-processing, including handling missing values, feature normalization, and encoding. The processed datasets were then concatenated to form a unified feature set, leveraging diverse information. Finally, feature scaling was applied to ensure uniformity, enhancing model performance and stability.

#### 2.4.1 Dataset Preprocessing

- `Data Concatenation`: The three datasets were combined by concatenating their respective features. This process yielded a final dataset with 350 features, which better represented the underlying patterns in the data.
- `Feature Naming`: To maintain clarity and consistency, all the features were renamed as feature_0 through feature_349.
- `Data Normalization`:A normalization technique was applied to standardize the data, ensuring each feature had a mean of zero and a standard deviation of one. This step was essential for optimizing model performance by scaling features to a comparable range.

#### 2.4.2 Analysis on Different Models Tried

Different models were evaluated on the mixed dataset to determine which yielded the best performance.

| **Model** | **Accuracy (%)** |
|---|---|
| Logistic Regression | 96.59 |
| SVM | 97.15 |
| Random Forest | 96.97 |
| LightGBM | 97.55 |

Table 4: Model Accuracies

#### 2.4.3 Performance Analysis on LightGBM

1. `Hyper-Parameters`:
   - `objective`: binary,
   - `metric`: binary_logloss,
   - `boosting type`: gbdt,
   - `learning_rate`: 0.1,
   - `num_leaves`: 31,
   - `feature_fraction`: 0.3.

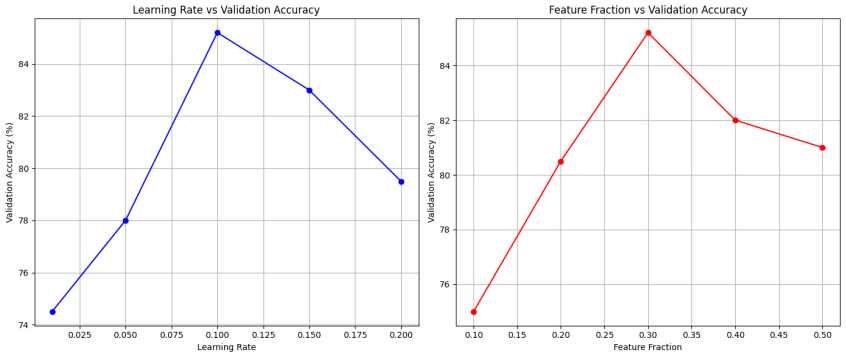



Figure 8: Validation Accuracy vs Learning Rate (*on left*) and Validation Accuracy vs Feature Fraction (*on right*)

2. `Training Data Usage vs. Validation Accuracy`: Initially, with smaller portions of the dataset (10%-30%), the model shows a rapid improvement in validation accuracy, suggesting that the model is learning key patterns effectively. Beyond 50% of the data, the accuracy stabilizes and exhibits marginal improvements, reaching a peak at **97.75% when trained with 60%** of data.

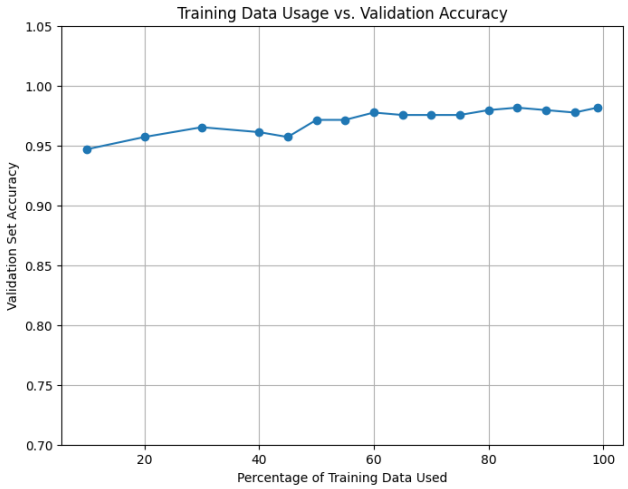



Figure 9: Plot for Training Data Usage vs Validation Accuracy

### 2.4.4 Final Results

- `Validation Accuracy`: 97.75%.
- `Confusion Matrix`:
  - Class 0: 246 correct predictions, 6 misclassified.
  - Class 1: 231 correct predictions, 6 misclassified.