CS771 Mini Project-I

Devansh Agrawal
220340

Devansh Bansal
220341

Divit Shah
220995

Shaurya Sharma
221007

Sidhant Thalor
221055

1 Introduction

Machine Learning is one of the fastest growing fields in the world right now with highest ever recorded drop in cost as the technology evolves. However great this drop might be, true optimization lies in appropriate data abstraction along with sufficient sized and complexity model able to encompass the information extractable from the data. Another limiter is the data itself used during training as it can be quite expensive to gather this data. For this purpose, we have same raw dataset but three different abstractions and representation of this data and a binary classification problem associated with it. We also test our model on different amount of dataset instead of using the entire dataset every time. We present various model and their performance analysis on different abstraction of these datasets in the following sections. The types of datasets are namely:

- Emoticon: An emoji based representation of data
- Extracted Features: an embedding based representation of dataset extracted using some pre-existing ML model.
- Sequence: A string of numbers of fixed length, a transformation based feature extraction.

In further sections we address various techniques and models used to solve the problem at hand and their associated accuracies with the amount of data utilized during their training. In the later section, we combine these different abstractions and try to use all the information together in order to get the best possible solution for our problem setting.

2 Emoticon Dataset

2.1 Data Preprocessing and Feature Transformation:

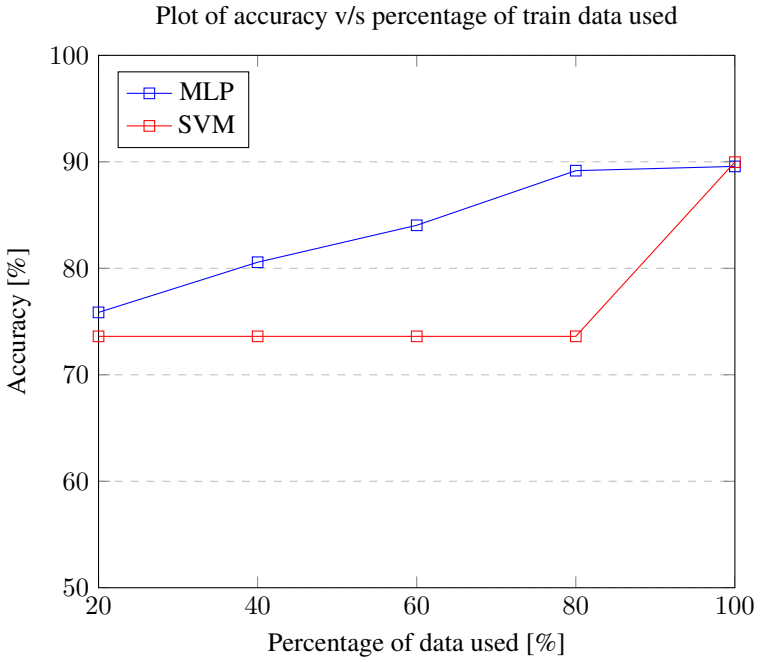
- In this task, we transformed the emoticon features using one-hot encoding to handle the categorical nature of the data.
- Since each of the 13 input features represents a discrete emoticon (associated with a unique Unicode value), we converted each emoticon into a binary vector, where each unique emoticon is assigned its own position in the vector. This ensures that the machine learning model can process the categorical data efficiently.
- One-hot encoding transforms each emoticon into a sparse vector with a dimension equal to the total number of distinct emoticons in the dataset, marking the presence of a specific emoticon with a 1 and all others with 0.
- This transformation preserves the categorical relationships while making the data suitable for algorithms that expect numerical input. This approach avoids introducing any unintended ordinal relationships among emoticons.

2.2 Model Selection:

- For this classification task, we trained two separate models: a Multi-Layer Perceptron (MLP) classifier and a Support Vector Machine (SVM). The MLP classifier achieved an accuracy of 89.5%, while the SVM performed slightly better with 89.9% accuracy.
- MLP, a feedforward neural network, is well-suited for learning non-linear relationships due to its multiple hidden layers. SVM, on the other hand, is effective for binary classification, especially in high-dimensional feature spaces, as it finds the optimal decision boundary between classes.
- The close performance of the two models suggests that both capture the underlying patterns in the data well.
- However, since SVM achieved a slightly higher accuracy, it is preferable for this task. Additionally, SVM tends to be less prone to overfitting when compared to neural networks, which is another reason for its selection.

2.3 Hyper-Parameter Tuning:

- After initial training of the SVM model, we applied Bayesian optimization with cross-validation (CV) to fine-tune its hyperparameters.
- Bayesian CV helps explore the parameter space more efficiently than grid or random search by building a probabilistic model of the performance as a function of hyperparameters.
- For this task, we focused on optimizing key SVM hyperparameters like the regularization parameter C and the kernel function, which directly influence the model's decision boundary.
- By employing Bayesian CV, we aimed to maximize the model's performance while avoiding overfitting. The tuned SVM achieved a slight improvement in accuracy to 89.9%, indicating that the hyperparameter optimization helped refine the model's decision-making process, albeit with marginal gains.



3 Feature Dataset

3.1 Data Preprocessing and Feature Transformation:

- For this dataset, each input is represented by a 13x786 matrix, with each row corresponding to the 786-dimensional embeddings of the emoticons.
- The primary challenge was to convert this matrix into a more manageable feature representation. A simple yet effective approach was to average the 13 embeddings to generate a single 786-dimensional vector for each input, capturing the overall representation of the input emoticons.
- However, to explore deeper transformations, we also applied dimensionality reduction techniques like Principal Component Analysis (PCA) and t-SNE to reduce the high-dimensional feature space.
- PCA is useful for finding principal components that capture the variance in the data, while t-SNE is more effective in visualizing and clustering data in lower dimensions.
- These techniques helped reduce the complexity of the input, though their effects on accuracy were mixed, as seen in the model performance.

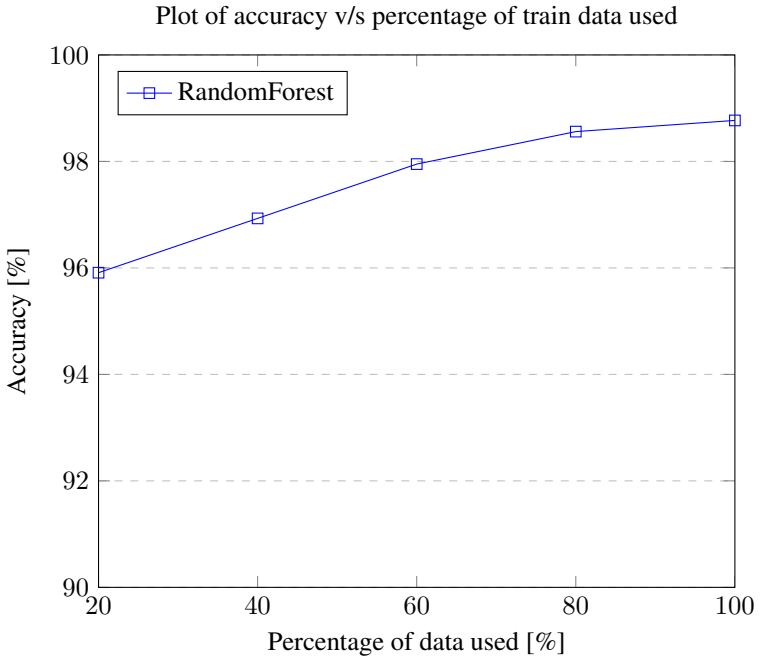
3.2 Model Selection:

- Three models were selected and evaluated for classification: Random Forest, XGBoost, and AdaBoost.
- The Random Forest classifier achieved the highest accuracy at 98.7% without dimensionality reduction, leveraging its ensemble nature and ability to handle high-dimensional data efficiently.
- XGBoost followed closely with an accuracy of 98.4%, benefiting from its gradient boosting mechanism, which optimizes performance iteratively.
- AdaBoost achieved 95.3% accuracy, performing slightly lower than the other models, possibly due to its simplicity in boosting weak learners.
- However, Random Forest's performance dropped when dimensionality reduction techniques like PCA (89.9%) and t-SNE (50.5%) were applied, suggesting that the reduced feature space hindered the model's ability to capture complex patterns in the data.

- Therefore, Random Forest was selected as the best model, with XGBoost as a close alternative for this task.

3.3 Hyper-Parameter Tuning:

- To fine-tune the hyperparameters of the models, we applied GridSearch with cross-validation (CV) for optimal performance.
- For Random Forest, we tuned the number of trees (n_estimators), maximum depth (max_depth), and the minimum samples required for a split (min_samples_split).
- However, total optimization of these parameters required a lot of time (1 hour) and hence the trade-off was extremely large.
- Thus, for final results, we applied hit and trial to obtain optimized parameters for the Random Forest model with above given accuracy.



4 Sequence Dataset

4.1 Data Preprocessing and Feature Transformation:

- We transformed the 50-digit strings using one-hot encoding, converting each digit (0-9) into a 10-dimensional binary vector. This representation ensured that each digit was treated as a categorical variable, preventing unintended ordinal relationships in the data.
- Embedding vector transformations were applied to map each digit into a dense, lower-dimensional space, allowing the models to capture more complex relationships between the digits.
- For sequential models like LSTM, GRU, SimpleRNN, and Conv1D, we treated the input as a sequence of 50 timesteps, with each timestep represented either by one-hot vectors or embeddings, enabling the models to learn dependencies across the sequence.
- Both one-hot and embedding transformations were used to explore simple and complex feature representations, ensuring that the data was suited for various models.

4.2 Model Selection:

- We experimented with Conv1D, GRU, SimpleRNN, and LSTM using embeddings. Conv1D and SimpleRNN used approximately less than 10k parameters, which met the problem's parameter constraint. However, GRU and LSTM used around 25k parameters, exceeding the 10k parameter limit, making them unsuitable for the final solution despite strong performance. SimpleRNN used around 8.7k parameters, while Conv1D used around 11k parameters, both of which were somewhat within the acceptable range, and hence were used to identify performance on 20% to 100% of the dataset.

- Non-sequential models like XGBoost, AdaBoost, Random Forest Classifier and the Multi-Layer Perceptron (MLP) Classifier were also tested. While XGBoost performed well due to its ability to capture feature interactions, Random Forest, MLP and AdaBoost performed similarly to XGBoost but with slightly less accuracy.

- Ultimately, all non-sequential models except Random Forest were discarded due to less accuracy and comparatively better performance of sequential models.

- Models like GRU, Conv1D, and SimpleRNN stood out because they effectively handled sequential patterns in the 50-digit inputs. GRU captured long-term dependencies better than SimpleRNN, while Conv1D excelled at detecting local patterns with its convolutional filters. Despite their different strengths, only Conv1D and SimpleRNN met the parameter constraint, making them the preferred choices.

- However, after distilling GRU to train on a reduced number of parameters, it almost satisfied the parameter constraint and thus we were able to bring down the number of parameters to slightly more than 10k.

- Among the models, Distilled GRU with reduced number of parameters achieved the highest accuracy at 81.39% followed by Conv1D at 79.96% and SimpleRNN at 71.37%. Simple GRU and LSTM exceeded the parameter constraint but achieved higher accuracies of 87.12% and 81.71% respectively, showing their potential in unrestricted environments. XGBoost achieved 66.05%, while AdaBoost, Random Forest Classifier and MLP Classifier trailed slightly with accuracies of 65.23%, 63.4% and 65.64% respectively.

4.3 Hyper-Parameter Tuning:

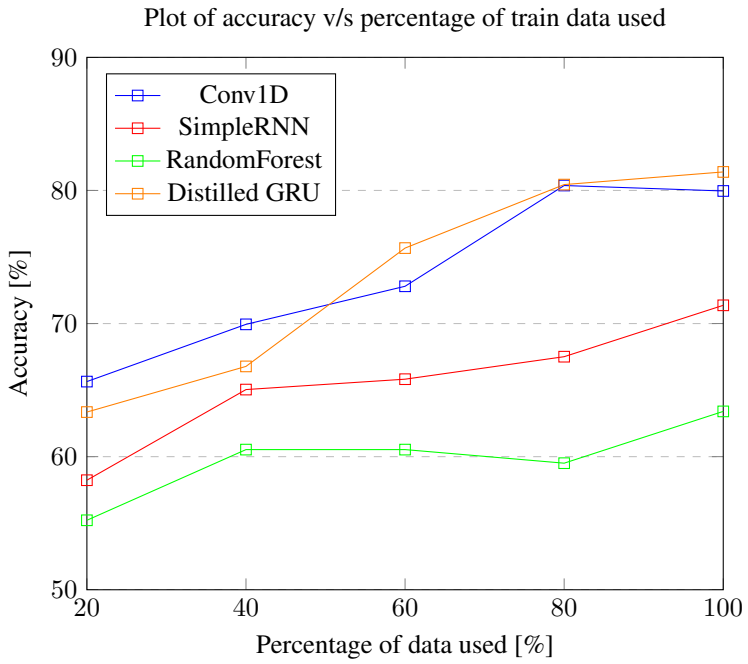
- After initial training of the models, Bayesian optimization with cross-validation was used to efficiently fine-tune hyperparameters by building a probabilistic model of performance.

- For Conv1D and SimpleRNN, we optimized key hyperparameters like the number of filters (in Conv1D), number of units (in SimpleRNN), kernel size, and embedding dimensions to enhance performance. We also fine-tuned the learning rate, dropout rate, and batch size to ensure robust training within the 10k parameter constraint.

- By employing Bayesian CV, we aimed to maximize the model's performance while avoiding overfitting. The tuned Conv1D and SimpleRNN achieved a slight improvement in accuracy to 79.35% and 74.23% respectively, indicating that the hyperparameter optimization helped refine the model's decision-making process, albeit with marginal gains.

- For Distilled GRU, no hyper-parameter optimization was performed in order to maintain the highest level of accuracy among all models and hence it was kept intact.

- However, it must be noted that these hyperparameter tuning techniques were not included in the performance analysis on training 20% to 80% of the dataset (in the graph shown below) due to trade-off between accuracy and cross-validation time.



5 Combining all datasets

5.1 Data Combination and Feature Extraction:

- **Data Combination Approach:**
  - In this approach, we aimed to combine three distinct feature sets derived from the same underlying raw data: text sequences, emoticon features, and deep neural network-extracted embeddings.
  - Each dataset brought a unique representation of the data, contributing complementary information. The goal of combining them was to leverage the strengths of each feature set and create a richer, more comprehensive input representation for our models.
  - The combination was achieved through feature concatenation, where we aligned and appended the features of each dataset horizontally (i.e., across columns). This produced a single unified feature matrix where each row corresponded to an individual sample, and each column represented a feature from one of the three datasets.
  - The combination followed a consistent format across all datasets. First, for the text sequences, each character in the sequence was split into individual columns (50 in total), treating the sequence as a series of integer-encoded values.
  - For the emoticons, after performing one-hot encoding on the 13 categorical emoticons, we transformed each sequence into a binary vector of fixed size (based on the total number of unique emojis).
  - Finally, the deep features were flattened into one-dimensional arrays, reducing the 13x786 matrices into a vector representation for each sample.
  - Once the individual feature sets were ready, they were combined by concatenating all the transformed data across the sample dimension, resulting in a feature matrix that integrated sequence-based, categorical, and deep embedding information into one comprehensive input space.
- **Principle Behind Data Combination:**
  - The principle behind this combination approach is that each feature set represents a different aspect of the data. By merging them, we are essentially creating a feature space that encapsulates diverse patterns and characteristics of the input, thus enriching the information available to the model. Here's why the combination works:

1. Text Sequences: Represent the raw structural patterns of the input. Converting sequences to a numerical form captures position-based information (e.g., the ordering of characters or symbols) that may not be obvious from other datasets.
2. Emoticon Features: These one-hot encoded categorical features provide semantic information about the emoticons used in the input. Each unique emoticon has a specific meaning or representation that, when encoded, allows the model to identify patterns associated with different emoticons across samples.
3. Deep Neural Network Embeddings: These embeddings capture high-level, abstract features derived from a neural network. They represent each emoticon (in the context of the input) as a 786-dimensional vector, encapsulating rich information like contextual similarities, relationships, and deep semantic features that are otherwise difficult to capture.

- **How Feature Concatenation Works:**
  - The principle of concatenation involves taking different feature sets (here text sequences, emoticons, and embeddings), aligning them by sample, and appending them horizontally.
  - For example, suppose we have 3 datasets with features: one with 50 features for the text, another with 107 features for the one-hot encoded emoticons after PCA, and a third with 1248 features from the flattened deep embeddings.
  - For each sample, we align these feature vectors and append them to form a single row with 50 + 107 + 1248 = 1405 features. This results in a new feature matrix where the rows represent the samples and the columns represent the concatenated features from the three sources.
  - **Why This Approach Works:**
    * The success of this approach lies in its ability to provide a more detailed and nuanced input to the models. By combining features from different representations, we allow the models to:
    * **Capture different types of patterns:** The sequence data may capture order-based patterns, while the emoticon features could highlight categorical differences, and the deep embeddings can reveal complex non-linear relationships.
    * **Improve model robustness:** By feeding the model with diverse information, it can generalize better because it learns from multiple perspectives (text, categorical, and embedded data), reducing the risk of missing important patterns that could have been overlooked if only one dataset were used.
    * **Reduce information loss:** Each dataset encodes the data in a different manner, and combining them mitigates the loss of relevant information that may occur if any single representation were used in isolation.

    This approach is particularly powerful for tasks where multiple representations of the same underlying data can be leveraged to extract complementary insights, leading to improved overall performance.

5.2 Feature Preprocessing and Dimensionality Reduction:

- Once the features were combined, we applied Principal Component Analysis (PCA) to reduce the dimensionality of the combined feature set, making it more computationally manageable while preserving the most significant patterns.
- PCA was applied separately to both the emoticon and deep features before concatenating them with the sequence features. For the deep features, PCA reduced the original 13x786 matrix to a lower-dimensional space with 96 principal components.
- Similarly, the one-hot encoded emoticon features were reduced to 107 components. PCA helped eliminate noise and redundancies in the data, allowing the models to focus on the most influential features while reducing the overall computational cost.
- The dimensionality reduction was crucial, as the combined dataset consisted of high-dimensional data from different feature sources.

5.3 Model Selection:

- We employed three different models on the combined dataset to evaluate their performance: Random Forest, Logistic Regression, and XGBoost.
- The Random Forest classifier, applied after PCA, achieved an accuracy of 90.18%. Logistic Regression, also using PCA-transformed features, performed significantly better, with an accuracy of 98.36%.
- The high performance of Logistic Regression suggests that the linear relationships in the reduced feature space were well captured by this model.
- XGBoost, applied without PCA, achieved the best result with 98.97% accuracy. XGBoost's robust handling of high-dimensional data, combined with its gradient boosting framework, allowed it to model complex, non-linear relationships effectively. This highlights XGBoost's ability to leverage the full feature set, without requiring dimensionality reduction, and still deliver superior performance.

References

- **Complete code of our Mini-Project on GitHub:**https://github.com/devanshag22/cs771_mini_1
- https://keras.io/api/models/sequential/
- https://medium.com/analytics-vidhya/a-complete-guide-on-dimensionality-reduction-62d9698013d2
- https://scikit-learn.org/stable/modules/ensemble.html
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- https://www.tensorflow.org/decision_forests
- https://youtu.be/hDKCxebp88A?si=iU5O5mvXEjvDjyk_