CS771:Mini Project-1 (Group 34)

Akshat Singh Tiwari 210094
Anuj Chaudhary 210167
Sai Vedant 210901
Sunny Raja Prasad 211078
Tanmey Agarwal 211098

October 22, 2024

1 Introduction

In this project, we were tasked with solving a binary classification problem. The challenge involved determining the best feature engineering and model selection techniques to achieve optimal performance. This report outlines the steps taken, from initial exploration of the dataset to the final model selection, and discusses the results obtained.

2 Approach for Dataset 1

For the first dataset, each input was represented as a sequence of 13 emoticons. In total, there were 214 unique emoticons present in the training data. This presented a unique challenge, as emoticons themselves do not have inherent numerical values that can be directly utilized in machine learning models. Therefore, various encoding and modeling techniques were explored to transform the emoticons into useful features for classification.

2.1 Feature Engineering

The initial approach was to count the frequency of each emoticon in the input sequence and use these counts as features. This method, however, resulted in poor performance. The main reason for this underperformance was likely due to the fact that counting frequencies failed to capture the sequential or positional information of the emoticons, which might have been important for the classification task.

To better represent the emoticons, a one-hot encoding scheme was used. Each emoticon in the sequence was represented as a one-hot vector of dimension 214 (since there were 214 unique emoticons in the training data). This transformed each input into a matrix of size 13×214, where each row corresponded to one of the 13 emoticons in the input. This representation allowed the models to treat each emoticon as a distinct categorical feature, preserving positional information while avoiding the shortcomings of frequency-based encoding.

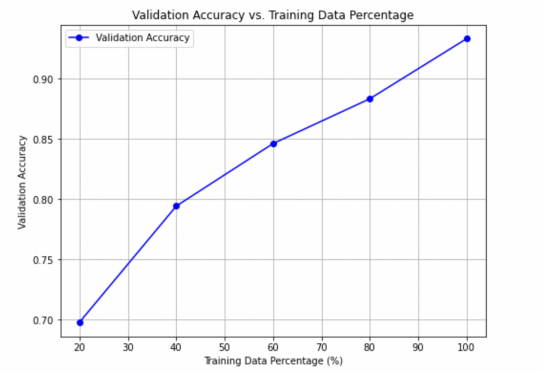
2.2 Model Selection

With the one-hot encoded and flattened features, several machine learning models were tested to determine the best approach for classification. We used the `scikit-learn` library for implementation.

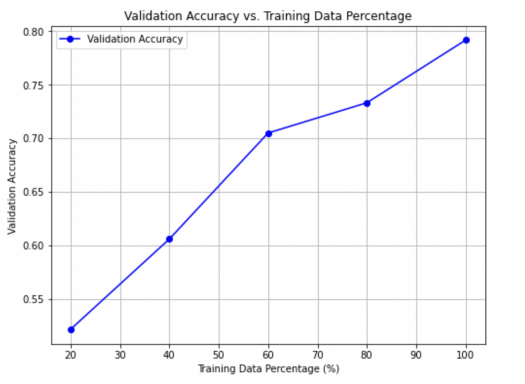
- **Logistic Regression**: Logistic regression was applied first due to its simplicity and efficiency with high-dimensional, sparse data. Given the linear nature of the one-hot encoded features, logistic regression performed well and was able to establish clear decision boundaries between the two classes.
- **Support Vector Machine (SVM)**: SVM was tried next, given its ability to handle high-dimensional spaces and complex relationships. However, the additional complexity of SVM did not result in significant improvements over logistic regression. The linear separability of the one-hot encoded features allowed logistic regression to outperform SVM.
- **Random Forest and Decision Tree**: Tree-based methods, such as Random Forest and Decision Tree, were also applied to capture potential non-linear relationships in the data. However, these models struggled with the high-dimensional and sparse nature of the one-hot encoded features. The splitting criteria in tree-based models did not effectively leverage the positional and categorical structure of the flattened vectors, resulting in poorer performance.

2.3 Results

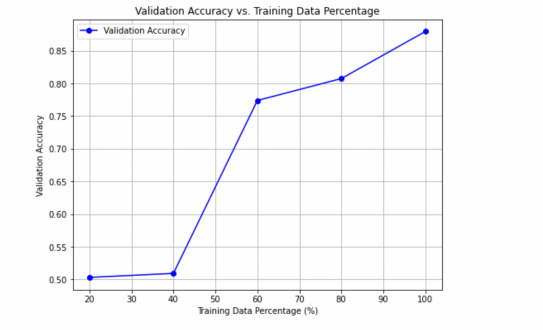
Logistic regression ultimately provided the best results for this dataset, outperforming the other models. This success can be attributed to the linear nature of the relationships between the one-hot encoded features and the target labels. SVM, while a strong competitor, did not show significant improvement over logistic regression, and tree-based methods struggled with the sparse nature of the data.



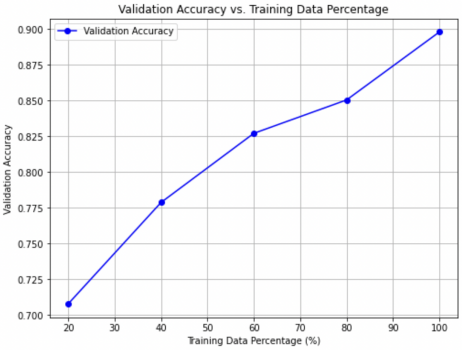
(a) Logistic Regression



(b) Decision Tree



(c) Random Forest



(d) Support Vector Machine

Figure 1: Accuracy vs. Fraction of Data for Different Models

3 Approach for Dataset 2

The second dataset represented each input as a matrix of size 13×768, where the rows corresponded to different features for each of the 13 input elements. Given the high dimensionality of this dataset, the challenge was to extract meaningful information while addressing potential overfitting and computational efficiency concerns.

3.1 Feature Engineering

To handle the high-dimensional matrix representation of the features, several approaches to dimensionality reduction and transformation were considered.

3.1.1 Vertical Compression

This approach aimed to compress the 13 rows (representing different input elements) by aggregating or summarizing them. However, this method led to a loss of critical information since each row contained distinct features for the respective input element. Consequently, vertical compression did not yield good results, as it eliminated important details necessary for classification.

3.1.2 Horizontal Compression

Horizontal compression was applied to reduce the number of columns (features) in the matrix, thereby preserving the structure of the input elements while reducing the complexity of each feature vector. This approach gave better results, as it retained the essential distinctions between different input elements while lowering the dimensionality. The improvement suggests that compressing the feature vectors helped reduce noise and irrelevant information, enhancing the model's ability to learn patterns.

3.1.3 Flattening

The matrix was then flattened into a vector of size $13 \times 768 = 9984$. This approach allowed the models to treat the input as a single, high-dimensional feature vector, rather than a structured matrix. Flattening provided the best results, likely because it allowed the models to capture both the input elements and their relationships within the entire feature set, without losing information through aggressive compression.

3.1.4 Dimensionality Reduction with PCA

After flattening the matrix, the next step was to reduce the feature space to avoid overfitting and improve model efficiency. Principal Component Analysis (PCA) was applied to the 9984-dimensional feature vector to reduce the number of features to 100 while retaining most of the variance.

PCA was able to reduce the dimensionality by capturing the most informative components, with a careful selection of the number of components based on the cumulative variance explained. This process helped in reducing computational costs while maintaining model performance.

3.2 Model Selection

Once the features were flattened, several machine learning models were tested to identify the most effective one for this dataset.

- **Logistic Regression**: Logistic regression was chosen as an initial baseline due to its simplicity and ability to work well with high-dimensional data. However, given the complexity and high feature count, logistic regression did not perform as well as expected. The model struggled to capture non-linear relationships within the flattened vectors.

- **Support Vector Machine (SVM)**: SVM emerged as the most effective model for this dataset. The ability of SVM to handle high-dimensional data and find a maximum-margin hyperplane between classes made it well-suited for this large, flattened feature space. SVM's performance improved significantly with this dataset, outperforming other models due to its robust handling of complex, high-dimensional feature sets.

- **Random Forest and Decision Tree**: These models were tested to capture non-linear relationships in the data. However, the performance of tree-based methods lagged behind SVM. The high dimensionality and sparsity of the flattened feature

vectors made it difficult for tree-based models to split the data effectively. Additionally, Random Forest and Decision Tree models were more prone to overfitting compared to SVM.

3.3 Results and Additional Insights

A key observation with this dataset was that even when trained with significantly less data, the models still delivered strong results. This suggests that the features in this dataset were particularly informative, allowing the model to generalize well despite the smaller training set. The structure of the 13×768 matrix, when flattened, provided enough discriminative power for the model to learn effectively from fewer samples.

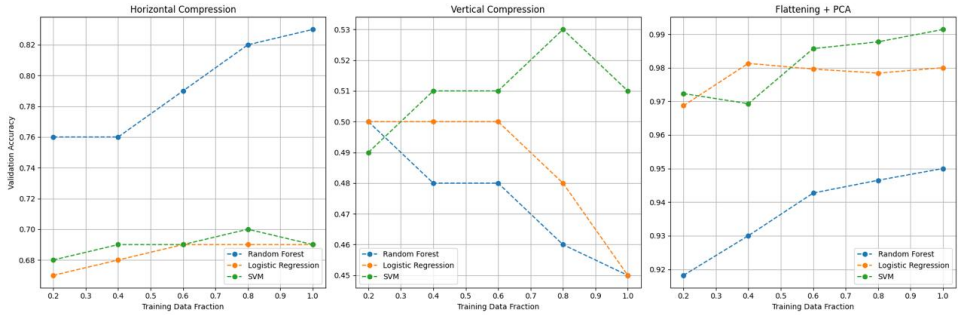


Figure 2: Performance comparison for Dataset 2

4 Approach for Dataset 3

The third dataset represented each input as a sequence of 50 digits, where each digit ranged from 0 to 9. This was transformed into a matrix of size 50×10, with each row corresponding to a one-hot encoded representation of a digit. Given the sequential nature and the relatively low dimensionality of the input, the challenge was to capture dependencies between the digits while balancing model complexity with parameter constraints.

4.1 Model Selection and Rationale

4.1.1 Traditional Machine Learning Approaches

Various machine learning models were initially tested using the `scikit-learn` library:

- **Logistic Regression**: The 50×10 matrix was flattened into a vector for use with logistic regression. This simple baseline model was unable to capture the sequential dependencies between digits, resulting in subpar performance.
- **Support Vector Machine (SVM)**: SVM was applied to the flattened vector, but while it performed slightly better, the lack of sequential awareness in the model limited its effectiveness.
- **Random Forest**: A random forest classifier was used to capture non-linear relationships in the data. However, it too struggled to exploit the sequential structure of the digits, yielding only modest results.

4.1.2 Shift to Deep Learning

Recognizing the limitations of traditional machine learning models, the focus shifted to deep learning. Convolutional Neural Networks (CNNs) were chosen due to their ability to effectively capture spatial and sequential patterns. TensorFlow was used to implement the CNN model, which operated directly on the 50×10 matrix.

4.2 CNN Model Architecture

The architecture was designed to exploit the structure of the input data and involved the following layers:

- **Input Layer**: The input shape was defined as $(50, 10)$ to accommodate the transformed sequence.
- **Conv1D Layer**: A 1D convolutional layer with 32 filters of size 3 was applied to capture local patterns across the sequence of digits.
- **Batch Normalization**: This was used to stabilize training and improve convergence by normalizing activations.
- **MaxPooling1D**: Two max pooling layers were applied to progressively reduce the dimensionality, retaining the most important features.
- **Fully Connected Layers**: Two dense layers with 32 neurons each were included to process the extracted features.
- **Output Layer**: A single neuron with sigmoid activation was used for binary classification.

The model was trained using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function.

4.3 Results

During experimentation, a more complex CNN model was tested and showed improved accuracy. However, due to a constraint on the number of trainable parameters (limited to under 10,000), a simpler architecture was selected. Despite these limitations, the model delivered strong performance with minimal overfitting, enhanced by the use of regularization techniques.

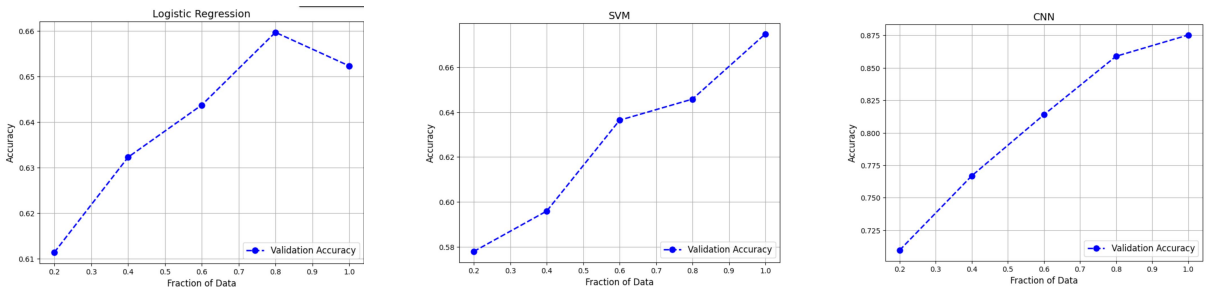


Figure 3: Three images without individual captions, arranged side-by-side.

5 Combined Dataset Approach

To leverage the strengths of all three datasets, the following steps were implemented:

5.1 Dataset Conversion and Creation

The datasets were combined into a new dataset by converting each to a suitable format. The first dataset was transformed into a $214 \times 13 = 9984$ length vector, the second into a one-dimensional vector, and the third, consisting of 50-length strings of digits from 0 to 9, was flattened to create a 500-length vector.

5.2 Dimensionality Reduction and Model Training

To address the high dimensionality of the combined dataset, Principal Component Analysis (PCA) was applied, effectively reducing the number of features while preserving variance. Models were trained using Support Vector Machines (SVM) and Logistic Regression on the new dataset, with the SVM model outperforming the Logistic Regression model in terms of accuracy.

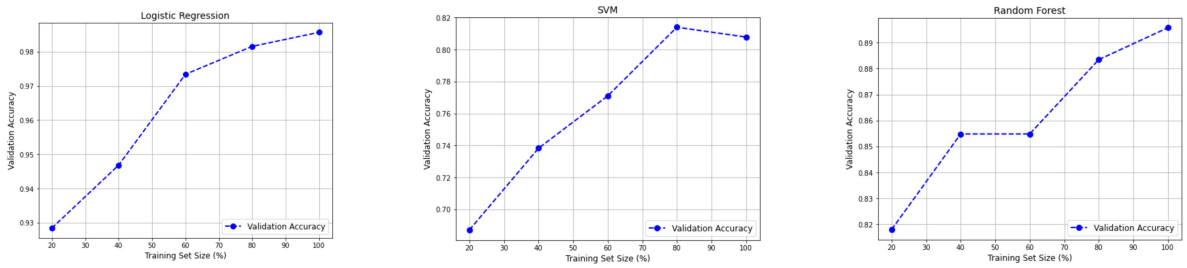


Figure 4: Three images without individual captions, arranged side-by-side.

6 Conclusion

The results indicated that the second dataset provided better accuracy compared to the combined model, even when using 100% of the training data. The reduced performance of the combined model could be due to the added complexity from integrating features across multiple datasets, which may introduce noise or redundancy, making it harder for the model to generalize effectively.

Dataset	**Model**	**Accuracy**
Emoticon	Logistic Regression	93.25 %
Features	SVM	99.12 %
Text Sequence, Col 1	CNN	87.14%
Combined	Logistic Regression	98.51%

Table 1: Comparison of model accuracies on validation set.

Figure 5: Final Comparision

7 References:

- **CS771:Lecture Notes**
- **Binary Classification Using Convolution Neural Network (CNN) Model.**
- **Hyperparameter Tuning in Random Forest.**