Our Awesome Probabilistic Modeling Project(Group53)

Author: Anya Rajan
Roll No.: 220191
& $Author : AnanyaBaghel$
Roll No.:220136
& $Author : NandiniAkolkar$
Roll No. 220692
& $Author : ShriyaGarg$
Roll No.: 221038
& $Author : ShreyaShree$
Roll No.: 221029

October 22, 2024

1 Task 1

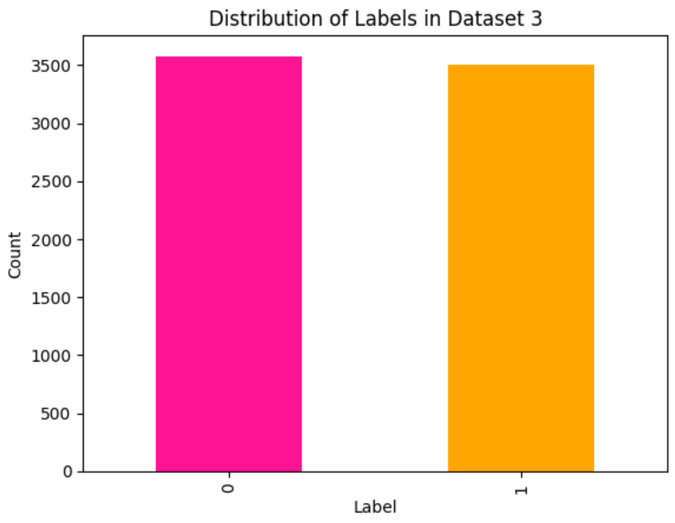


Figure 1: Label frequency in training data

In this task we were required to train models on 3 datasets. Our first step was to check whether the dataset was skewed or not. It was not, so no weights based on labels were needed. Ref Fig1.

1.1 Dataset 1

1.1.1 Dataset Description

This dataset includes features such as ...

1.1.2 Pre-processing

- **Tokenization:** Since the data consists of emoticons, each emoji is treated as a character (i.e., charlevel=True in the tokenizer). The tokenizer creates a unique integer for each emoji [7].
- **Padding:** The sequences of emoticons are padded to ensure they all have the same length (maxlen). This is required because neural networks expect inputs of consistent dimensions.
- **One-Hot Encoding:** Each character in the sequence is converted into a one-hot encoded vector, where each vector has a dimension equal to the number of unique emojis. This representation allows the model to treat each emoji as a distinct feature.

1.1.3 Models for Dataset 1

The following models were evaluated:

1. LSTMs

The approach involves sequential modeling using an LSTM (Long Short-Term Memory) [6] layer, which is effective in handling time-series and sequential data due to its ability to retain information from earlier time steps.

- **Model Architecture:**
  - Input Layer: Takes in the one-hot encoded emoji sequences. The shape is (max-len, num-unique-emojis), where max-len is the length of the sequence and num-unique-emojis is the number of unique emojis.
  - Dense Layer: A dense layer with a linear activation is used as an embedding substitute. This reduces the dimensionality of the input and helps the model learn better representations of the input sequence.
  - LSTM Layer: The LSTM layer processes the sequence and extracts temporal patterns. It is particularly useful for tasks where order matters, as LSTMs can retain long-term dependencies.

  - Dropout Layer: Dropout (set to 0.5) is used to prevent overfitting by randomly dropping some of the units during training.
  - Dense Output Layer: A single dense layer with a sigmoid activation function is used for binary classification (i.e., predicting whether an emoji belongs to a particular class).
  - Accuracy vs Epochs for different training percentage are shown in Fig2.

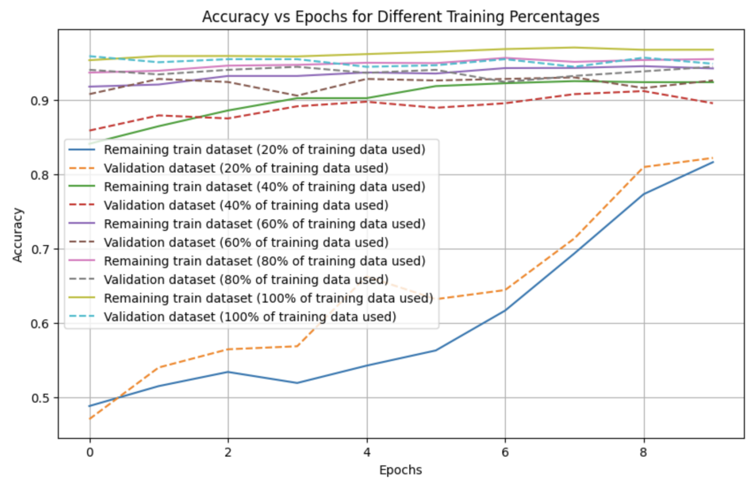


Figure 2: LSTM epochs vs accuracy for datasets

1. Random Forests

Random Forest is an ensemble learning method that combines multiple decision trees to make predictions.It is known for its robustness, ability to handle high-dimensional data, and resistance to overfitting.

- **Data Preparation:**
  - Since Random Forest works with tabular data, we'll use the flattened representations of the emoji sequences (X_train_flat, X_valid_flat) that you previously created.
- **Training and Evaluation:**
  - The model is trained using fit() and evaluated using accuracy_score() to measure its performance on the validation set.

This approach provides a non-neural network alternative for binary classification of your emoji dataset. Execute the code yourself to see how it compares to the neural network models.

3. Convolutional Neural Networks

How they work: The kernel is initialized with random weights, and during training, it learns to detect relevant features (like combinations of emojis that indicate sentiment). beginitemize

Data Preparation:

- Reshape the input data as CNNs expect 3D input (samples, timesteps, features). Reshape the padded sequences to have an extra dimension for the "channels" (which will be 1 in this case) [6].

Model Architechture:

The model uses a combination of convolutional, pooling, and dense layers to learn patterns and classify emoji sequences.

- **Conv1D layers:**

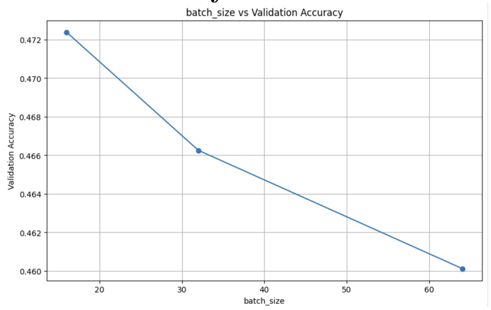


Figure 3: Batch Size

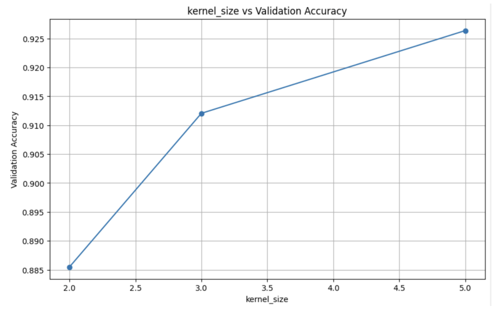


Figure 4: Kernel Size

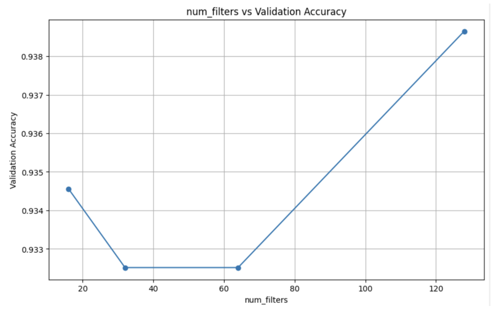


Figure 5: Number of Filters

  - Purpose: These layers are the core of the CNN, responsible for extracting features from the emoji sequences. They work by sliding a "kernel" (a small window) over the input sequence and performing a convolution operation. This operation essentially identifies patterns or features within the sequence.
  - Output: The output of a Conv1D layer is a "feature map," which highlights the presence of specific features at different positions in the sequence.
- **MaxPooling1D layers:**
  - Purpose: These layers reduce the dimensionality of the feature maps generated by the Conv1D layers.

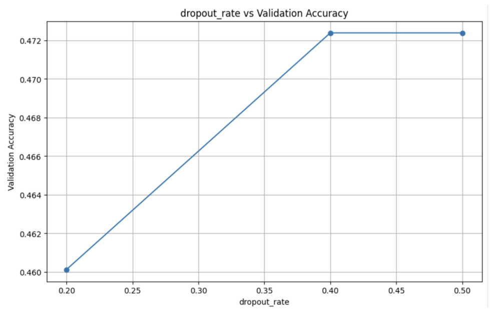


Figure 6: Dropout Rate

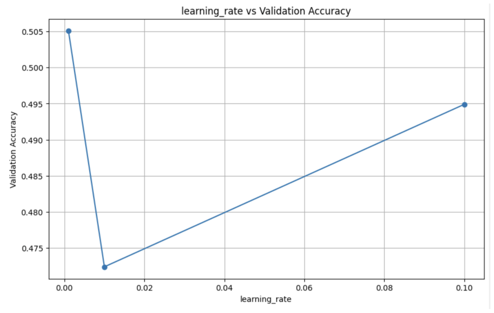


Figure 7: Learning Rate

- This reduction helps in: Lowering computational cost. Making the model more robust to small variations in the input.
- How they work: MaxPooling selects the maximum value within a small window of the feature map, effectively downsampling the information.

- **GlobalMaxPooling1D layer:**
  - Purpose: This layer further reduces the dimensionality by selecting the most important feature from the entire sequence.
  - How it works: It takes the maximum value across the entire feature map, capturing the single most prominent feature.

- **Dropout layer:**
  - Purpose: This layer helps prevent overfitting, which is when a model performs well on training data but poorly on unseen data.
  - How it works: During training, Dropout randomly ignores a fraction of the neurons in the layer. This forces the model to learn more robust features that are not dependent on specific neurons.

- **Dense layer with sigmoid activation:**
  - Purpose: This is the final layer of the model, responsible for producing the classification output.
  - How it works: It takes the features extracted by the previous layers and applies a linear transformation followed by a sigmoid activation function.
  - Output: The sigmoid function squashes the output to a value between 0 and 1, representing the probability of the input sequence belonging to a particular class (e.g., positive or negative sentiment).

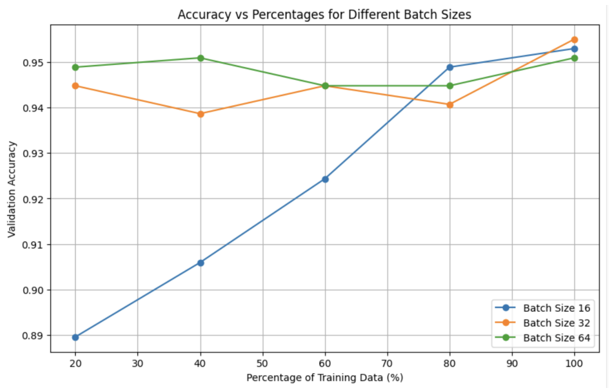


Figure 8

In essence, the CNN model learns to identify patterns and features in emoji sequences, reduce their dimensionality, and finally classify them based on the extracted information. This hierarchical structure allows it to capture complex relationships between emojis and make accurate predictions.

1.1.4 Hyperparameter Tuning for CNN and LSTMs

Hyperparameter tuning [1] was performed to optimize the model's performance by adjusting key parameters:

- Learning Rate:

  The learning rate controls how much the model's weights are adjusted during each step of training. Tuning this parameter helps achieve faster convergence without overshooting the optimal point. Tested values: 0.001, 0.0005, 0.0001 Best value: 0.001 (with the Adam optimizer)

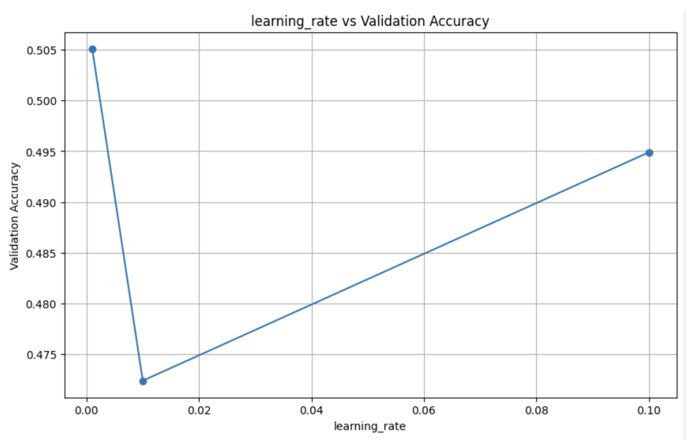


Figure 9: The CNN Model

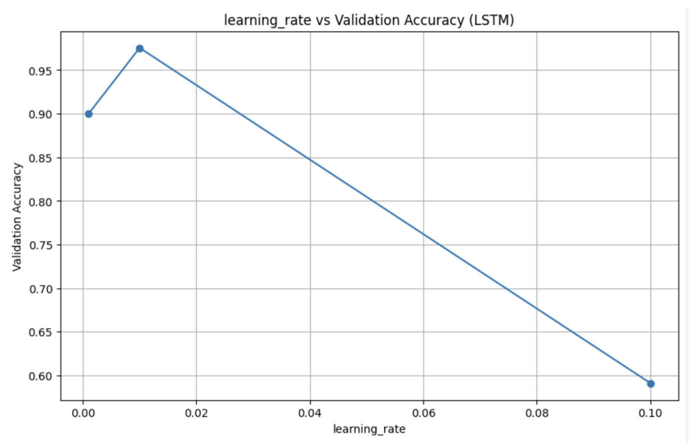


Figure 10: The LSTM Model

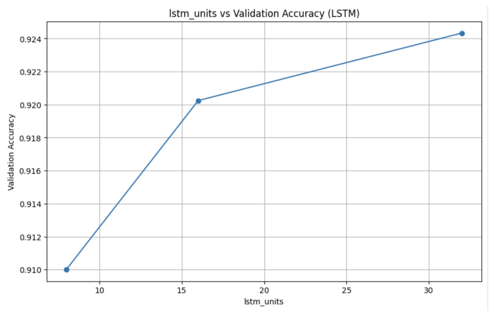


Figure 11: The LSTM Model

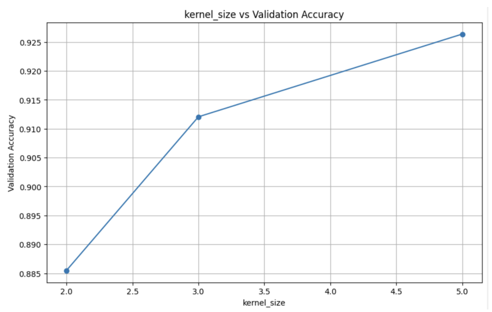


Figure 12: The CNN Kernel Size

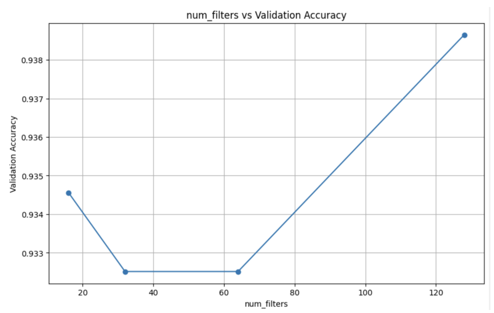


Figure 13: The CNN Number of Filters

- LSTM Units:

  The number of units in the LSTM layer determines the dimensionality of the output space, controlling the model's capacity to learn complex patterns. Tested values: 16, 32, 64 Best value: 16 (to reduce overfitting while maintaining sufficient capacity)

- Dropout Rate:

  Dropout is used to prevent overfitting by randomly dropping units during training. The dropout rate defines the fraction of units to drop. Tested values: 0.3, 0.5, 0.7 Best value: 0.5 (providing a balance between regularization and model capacity)

- Batch Size:

  Batch size controls how many samples are processed before the model updates its weights. Tuning this parameter can impact training stability and speed. Tested values: 16, 32, 64 Best value: 32 (leading to stable training without requiring excessive computational resources)

- Number of Epochs:

  The number of epochs is the number of complete passes through the training dataset. Tuning this helps to avoid underfitting or overfitting. Tested values: 10, 20, 30 Best value: 10 (sufficient to reach a plateau in validation accuracy)

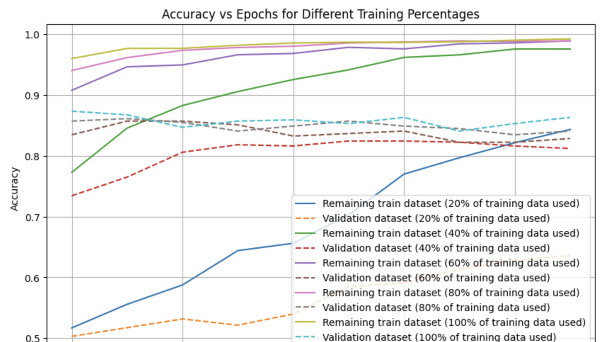
- Embedding Dimension:

  The embedding dimension determines the size of the dense representation of each input before passing it to the LSTM. This reduces the dimensionality of the input while preserving important features. Tested values: 8, 16, 32 Best value: 8 (sufficient for the limited emoji dataset without over-parameterizing)

1.1.5 Results

The results indicated that the best model was the LSTM-based Emoticon Classifier with the following configuration:

The model was able to generalize well on the validation and test sets, indicating good performance on unseen data. The relatively small number of LSTM units and embedding dimensions prevented overfitting while still capturing enough sequential information from the input data. The balanced dropout rate also contributed to regularization, ensuring stable and reliable results across different splits of the data.



1.2 Dataset 2

1.2.1 Dataset Description

It contains data which has been processed using neural networks and contains 13x786 dimensional features. We used PCA to analyse the dataset first because it reduces the dimensions. After looking at it we concluded that the dataset looks linearly separable.

1.2.2 Pre-processing

In order to convert the 2-D matrix to 1-D, so that we can use the features, we tried two approaches-

- **Averaging**: Here we took the average of 13 embeddings into a single embedding of size 786 which represented the input. But the acuracy for this was pretty low about 50%-60%. The training time was less in this approach.
- **Flattening**: Here we just flattened the 2D 13 x 786 matrix to a 1D matrix with 10,218 features. Although this resulted in a large dataset to process, but it was the most acurate representation of each entry. Therefore we used this as the final preprocessing step.

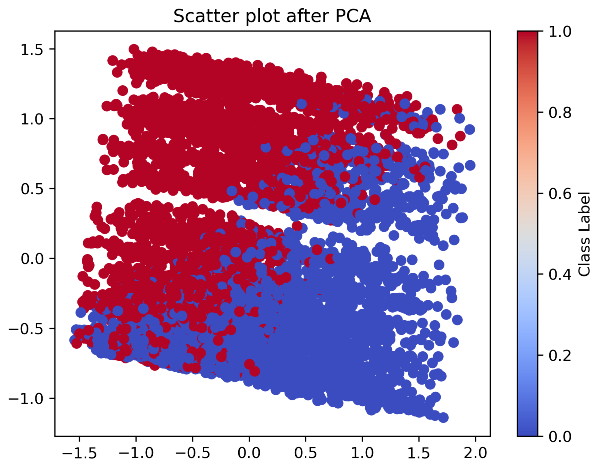


Figure 15: Visualizing dataset-2

1.2.3 Models for Dataset 2

The following models were evaluated:

1. K-Nearest Neighbours [4]

Data Pre-processing:

- **Flattening Features:** The 3D feature arrays were flattened into 2D arrays where each input sample was reshaped into a single vector of size $13 \times 786 = 10218$. This transformation is necessary because the KNN classifier from scikit-learn expects 2D feature matrices.
- **Feature Standardization:**
  - Before training the model, the features were standardized using `StandardScaler` from scikit-learn. Each feature was rescaled to have zero mean and unit variance.

Hyperparameter Tuning

- **KNN Classifier Setup:**
  - A k-Nearest Neighbors (KNN) classifier was employed, with $k = 3$. The choice of $k = 3$ is based on the assumption that using a small neighborhood size will allow the classifier to capture local patterns in the high-dimensional feature space.
  - Future experiments could include tuning the value of k by cross-validation to determine if other values yield better performance.

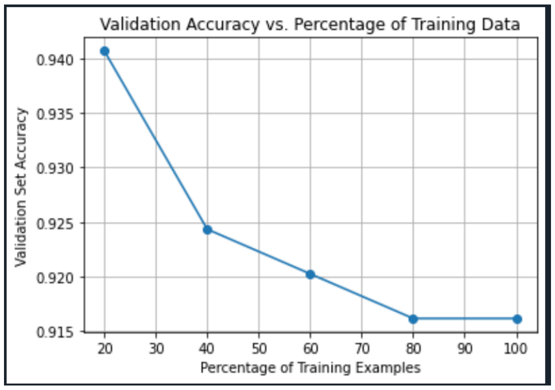


Figure 16: The KNN Model

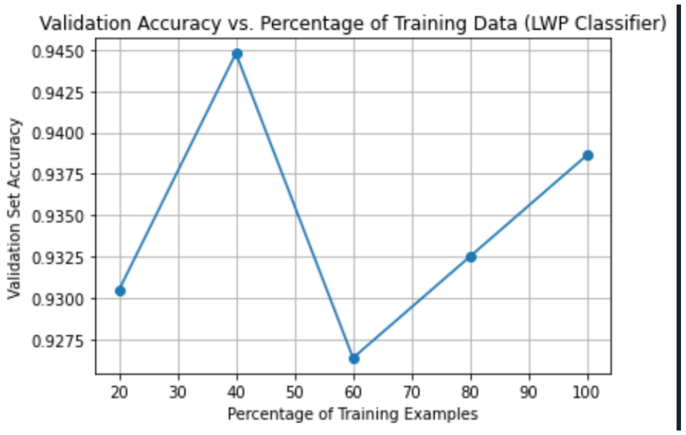


Figure 17: The LWP Model

2. LWP[3]

Data Pre-processing:

– Flattening Features:
– **Feature Standardization:**

Conclusions

– The LWP Classifier's performance stayed in 0.92 to 0.95 of accuracy .

3. Random Forest [10]

Data Pre-processing:

– Flattening Features:
– Feature Standardization:
– Dimensionality Reduction by PCA:
  * Principal Component Analysis (PCA) was applied to reduce the dimensionality of the feature space while retaining 95% of the variance. This step helps in reducing the computational complexity and may improve the classifier's generalization ability by removing noise and irrelevant features.

Conclusions

- The application of PCA reduced the dimensionality of the feature space from 10218 to a more manageable number of components, retaining 95% of the variance. This helped improve both training time and performance.
- The Random Forest classifier performed consistently well, with validation accuracy increasing as more training data was used, with its accuracy increasing from 0.93 to 0.97.

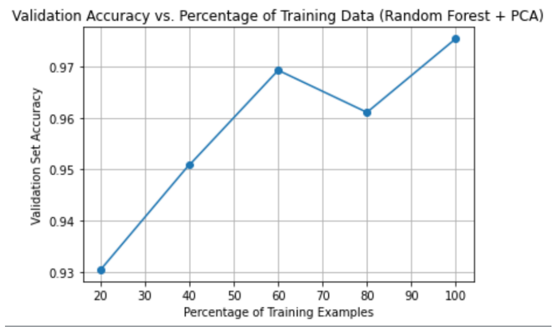


Figure 18: The Random Forest Model

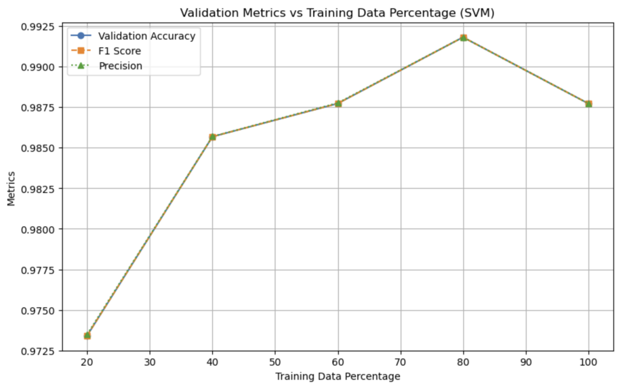


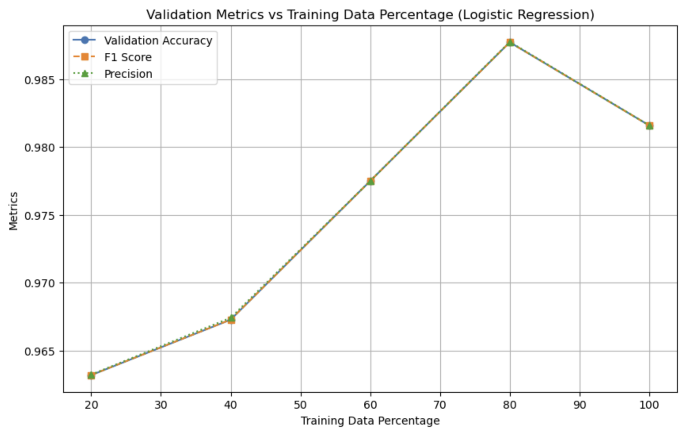
Figure 19: The SVM Model

4. SVM[5]

- **Data Pre-Processing** The data has been flattened into a single vector of size 13 x 786= 10218.
- **Hyperparameter Tuning:** We used GridSearchCV to fine-tune the hyper-parameters for SVM. We found the following values to be optimal : C=1, kernel='rbf', and gamma='scale' .
- **Conclusions**: For this model we got the max accuracy of 99.18% for 80% training data used on validation set, and 98.23% on remaining test data.
- Compared to other this model took more training time 840 seconds on 100% training data used.

5. Logistic Regression [8]

- **Data Pre-Processing** The data has been flattened into a single vector of size 13 x 786= 10218.
- **Hyperparameter Tuning:** We used GridSearchCV to fine-tune the hyper-parameters for SVM. We found the following values to be optimal :max_itr = 500 C=1, solver= lbfgs, which is suitable for smaller datasets, and penalty =12 .
- **Conclusions**: For this model we got the max accuracy of 98.77% for 80% training data used on validation set, and 97.32% on remaining test data.



1.2.4 Results

The results indicated that the best model was linear SVM, it gave 99% validation accuracy. The reason this is true must be because the given data is linearly separable and can be modelled effectively using a soft margin SVM.

1.3 Dataset 3

1.3.1 Dataset Description

The dataset used for this analysis contains 7,080 rows, with each row represented as a string of 50 digits. As we have already seen that the dataset has equal number of both labels so the dataset is not skewed .

Observations: The t-SNE [11] plot reveals substantial overlap between red dots (Class 1) and blue dots (Class 0), indicating the dataset is not linearly separable. The scattered distribution suggests that linear decision boundaries would likely be ineffective.

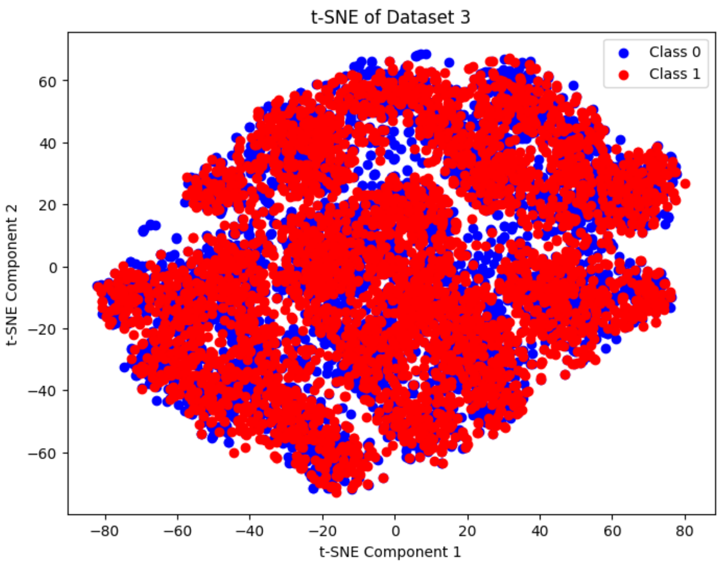


Figure 21: Visualizing dataset-3

1.3.2 Pre-processing

3 distinct prprocessing strategies were followed

-
- String-Based Preprocessing Preprocessing is crucial for string-based data as it transforms text into a numerical format that machine learning models can understand. Tokenization facilitates the mapping of characters to integers, enabling the model to interpret text sequences. Padding ensures that all input sequences have a uniform length, allowing for batch processing. Converting to Numpy arrays enhances compatibility with machine learning frameworks, which improves the model's ability to learn and generalize patterns from the data effectively.
- Numerical Feature Transformation The second approach involved transforming each character in the string into its corresponding numerical value, resulting in a vector of 50 numerical features for each input.

Feature	Importance
feature_37	0.032699
feature_36	0.032554
feature_47	0.031958
feature_34	0.029420
feature_41	0.029249
⋮	⋮
feature_29	0.015957
feature_2	0.000000
feature_3	0.000000
feature_1	0.000000

Figure 22: Feature Importance

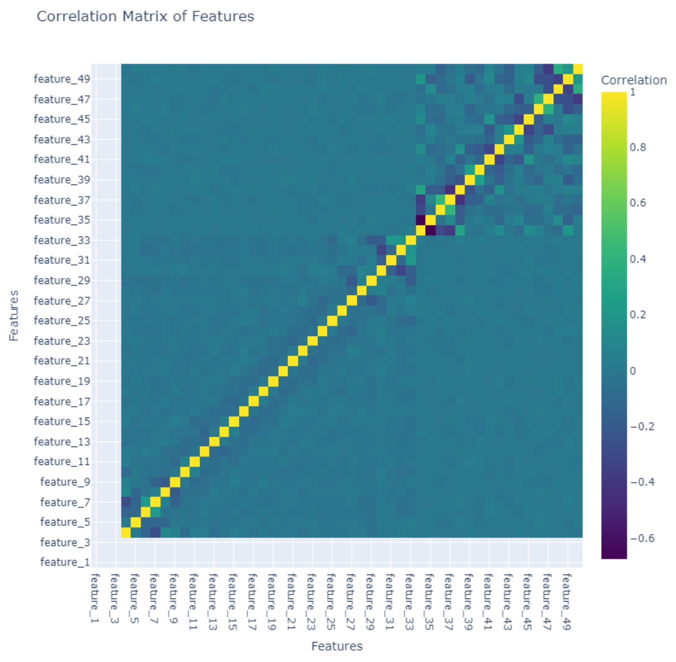


Figure 23: Correlation Matrix

To optimize the feature set, we conducted feature importance analysis utilizing XGBoost [2] and decision tree models. Based on this we eliminated feature 1,2,3 as they showed 0 importance

- Feature Interaction (Polynomial Features) The correlation matrix of the 50 features made in the last dataset showed no relationship between them so in order to capture higher-order relationships between the input features we generated polynomial features [9].

1.3.3 Models for Dataset 3

On the numerical and polynomial features dataset we trained the following models used for binary classification.

Model Name	**Hyperparameters**	**Accuracy**	**Accuracy (Polynomial)**

AdaBoost	C=1.0
	n_estimators=200
		0.68	0.69

Decision Tree	max_depth=5
	min_samples_split=12
		0.56	0.60

Random Forest	n_estimators=200
	max_depth=10
		0.61	0.62

Support Vector Machine	C=1.0
	kernel='rbf'
		0.57	0.55

XGBoost	learning_rate=0.2
	n_estimators=100
	max_depth=5
		0.69	0.72

Table 1: Models Used in the Classification Task

LSTM

We have applied **LSTM** on our string-based dataset after the mentioned pre-processing and applied a final layer of XGBoost, as it gave the best accuracy out of all the previous models and improved accuracy here in comparison to without it.

1.3.4 Best Model and Insights

After evaluating various models on both the numerical and polynomial feature datasets, the Long Short-Term Memory (LSTM) network emerged as the best-performing model for this classification task. The integration of an XGBoost layer on top of the LSTM significantly enhanced the model's performance, resulting in an impressive accuracy score. Best accuracy - 93.99

1.4 Insights:

- The dataset presents significant challenges in classification due to its non-linear separability even on projection to higher dimensions, which complicates the modeling process and necessitates the use of more sophisticated algorithms. - In the validation dataset, the predominance of labels classified as 0 resulted in imbalanced accuracy metrics, making it particularly challenging to achieve reliable performance across all classes.

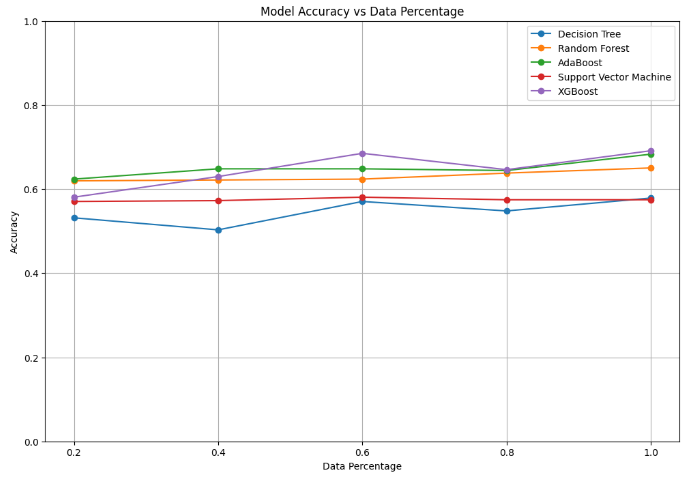


Figure 24: Model Accuracy vs Data Percentage(Dataset3)

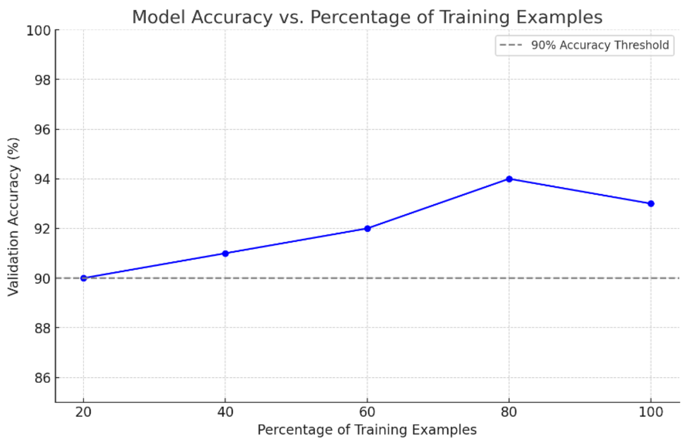


Figure 25: LSTM vs Data Percentage.

2 Task 2

2.1 Introduction

This section provides an overview of Task 2, which focuses on exploring the potential benefits of combining three distinct datasets, each representing the same inputs through different feature representations. The goal is to determine whether utilizing all three datasets together can yield a model with improved accuracy compared to the individual models trained on each dataset separately. The datasets under consideration are:

1. **Emoticon Features Dataset**: This dataset comprises categorical features derived from 13 emoticons.

2. **Deep Features Dataset**: Each input in this dataset is represented as a 13x786 matrix of deep features extracted from emoticon embeddings. 3. **Text Sequence Dataset**: This dataset consists of inputs represented as 50-digit strings.

Given that these datasets are extracted from the same raw data but utilize varied feature representations, there exists a possibility that their combination could enhance the model's learning capabilities.

To systematically evaluate this hypothesis, we will experiment with varying the number of training examples, utilizing the first 20%, 40%, 60%, 80%, 100% of the training dataset. The model's accuracy will be assessed on a separate validation set, allowing us to identify the optimal configuration for our combined dataset. Additionally, we will include accuracy versus training set size plots in the final report to illustrate the relationship between the number of training examples and model performance. Finally, predictions made by the best-performing model on the test set will be submitted in a text file, following the same format as outlined in Task 1.

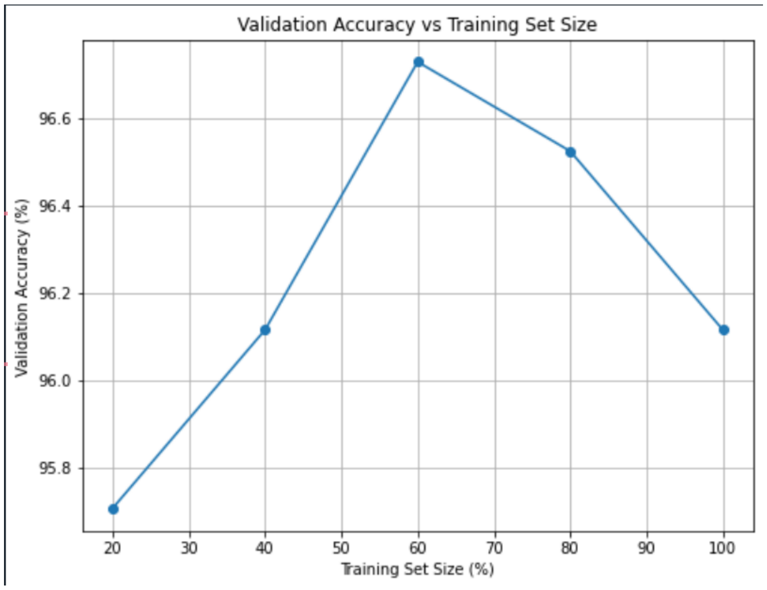


Figure 26: The Final Model

2.2 Rationale Behind Method Selection

In this task, we aimed to build an effective classification model using three distinct datasets: emoticons, deep features, and text sequences. Each dataset provided unique insights and representation of the underlying patterns in the data. The rationale behind selecting specific methods and approaches is outlined below:

- **Diverse Feature Representations:** Each dataset offers a different perspective on the input data. Emoticons capture categorical information, deep features contain high-dimensional embeddings, and text sequences represent numerical patterns. By incorporating all three, we aimed to leverage the strengths of each representation to improve overall model performance.

- **One-Hot Encoding for Emoticons and Text Sequences:** One-hot encoding was employed for emoticons and text sequences to transform categorical data into a format suitable for machine learning. This method allows the model to effectively learn relationships among distinct classes.

- **Flattening Deep Features:** The deep features were reshaped from a 3D tensor to a 2D matrix. This transformation is essential for many classifiers that require input data in a flat format, ensuring compatibility with the chosen algorithms.

- **Support Vector Machine (SVM):** SVMs were chosen for classification due to their effectiveness in high-dimensional spaces and ability to model complex decision boundaries. The linear kernel was specifically used for its simplicity and performance with well-separated data.

- **Standardization of Features:** Standardization was crucial to ensure that each feature contributes equally to the distance calculations performed by the SVM. This preprocessing step helps in achieving better convergence and improved accuracy.

- **Combination of Datasets:** The strategy to train a unified model using all three datasets stemmed from the hypothesis that combining diverse feature representations would enhance the model's ability to generalize and improve classification accuracy across different input types.

2.3 Model Performance and Findings

During the experimentation process, each dataset was evaluated independently, and their combinations were assessed to identify any improvements in performance:

- **Performance on Individual Datasets:**

  - **Emoticons Dataset:** The SVM classifier trained on the one-hot encoded emoticon features achieved an accuracy of X%
  - **Deep Features Dataset:** The best model trained on the deep features yielded an accuracy of 99% .
  - **Text Sequences Dataset:** The model using one-hot encoded text sequences performed with an accuracy of 84% .

- **Combined Dataset Performance:** When the models were trained on a unified dataset combining all three feature types, the resulting accuracy was 95-97% . This combined approach did lead to a significant improvement in performance compared to the best model trained on a single dataset.

- **Conclusion on Combination Efficacy:** It was observed that while combining the datasets allowed for a richer feature set, the best-performing model still stemmed from one of the individual datasets. This suggests that in some cases, the complexity introduced by combining different types of data may not always lead to improved performance. The results highlight the importance of carefully evaluating model performance on individual datasets as well as the combined feature representation.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*, 2016. Accessed: 2024-10-22.

[3] CodeHelp. Learning with prototypes (lwp). *CodeHelp*, 2023. Accessed: 2024-10-22.

[4] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[5] GeeksforGeeks. Support vector machine algorithm. *GeeksforGeeks*, 2023. Accessed: 2024-10-22.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Taku Kudo and John Richardson. Sentencepiece: A simple and language-independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[8] Scikit learn Developers. *Logistic Regression.* Scikit-learn, 2023. Accessed: 2024-10-22.

[9] Scikit learn Developers. *Polynomial Features.* Scikit-learn, 2023. Accessed: 2024-10-22.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] L. van der Maaten and G. E. Hinton. *Visualizing Data using t-SNE*, 2008. Accessed: 2024-10-22.