Our Awesome Probabilistic Modeling Project

Kushagra Srivastava
24111041
Kumari Ritika
241110039
Suvam Mukhopadhyay
241110074
Shubhashish Shukla
241110069
Krishanu Ray
241110037

October 22, 2024

1 Introduction

Probabilistic modeling plays a crucial role in handling uncertainty in machine learning tasks. This project focuses on binary classification, aiming to develop models that not only achieve high accuracy but also require a minimal amount of training data. We work with three distinct datasets, each representing different feature representations derived from the same raw dataset. The key objectives are:

- To train and evaluate binary classification models on each of the three datasets individually.
- To identify the best-performing models based on validation accuracy and training data efficiency.
- To explore the potential of combining different feature representations to improve model performance.

The project is divided into two main tasks: training individual models on each dataset and investigating the performance of combined feature representations.

2 Methodology

In this section, we outline the implementation details of the final models selected for each of the feature datasets. This includes a discussion on data preprocessing, feature extraction methods, and the training and validation processes used to assess model performance. We also provide a detailed description of the models that were chosen for the final implementation. The selection of these models was based on a set of criteria, including their ability to meet performance benchmarks such as accuracy and robustness, as compared to other models we explored during experimentation. By evaluating training and validation results, we ensured that the models incorporated in the final implementation were the most effective for each respective dataset.

Task 1:

2.1 *Emoticons as Features Dataset*

2.1.1 Data Preprocessing

- Since we are given emoticons, which are categorical in nature, we decided to apply One-Hot Encoding to represent them.
- To perform this, the code loads the Emoticons dataset and splits the emoticon strings into 13 separate columns, each storing a single emoticon for better feature handling.
- It applies **One-Hot Encoding** to convert categorical emoticon features into a numerical format suitable for training our models.
- We need to store the encoder because we have to apply the same transformations on validation/test datasets as well during validation/testing phase.

2.1.2 Training and Validation

- A loop iterates through different percentages of the training data (20%, 40%, etc.), allowing us to observe how the model's performance scales with more data.
- For each subset, the code:
  - **Encodes** the subset of training data using One Hot Encoding.
  - **Applies** the same encoder transformations on the validation data.
  - **Trains** a classifier using the training data subset.
  - **Evaluates** its performance on the validation set.
  - **Prints** the validation accuracy, enabling us to plot and analyze the relationship between training data size and model accuracy.

2.1.3 Model Description

- Our code trains 3 different models from scikit-learn. They are as follows -
  - **Logistic Regression:** Default version of Logistic Regression, which uses and L2 Regularizer and lbfgs solver was used
  - **Support Vector Machine:** SVM with a linear kernel was used
  - **Multi Layer Perceptron:** An MLP classifier was built using 2 hidden layers, each with 100 nodes each, was used
- Finally the plots are generated based on the accuracies and percentage of training data

2.2 *Deep Features Dataset*

2.2.1 Data Preprocessing

- **Loading Data:** The training and validation datasets are loaded from .npz files containing features and labels.
- **Flattening input:** Since each input is represented as a matrix, we decided to flatten out the matrix into an 1D array
- **Applying PCA** After flattening of input, the number of features become too large, so we decided to apply PCA and reduce the features into 100 components. 100 components was chosen after experimenting with multiple possible components values
- **Store the PCA transformation:** The PCA transformations we had applied on training data needs to be applied on the validation/ test datasets as well, so we need to store the transformation.

2.2.2 Training and Validation

- A loop iterates through different percentages of the training data (20%, 40%, etc.), allowing us to observe how the model's performance scales with more data.
- For each subset, the code:
  - **Applies** the PCA tranformations on the subset data.
  - **Applies** the same transformations on the validation data.
  - **Trains** a classifier using the training data subset.
  - **Evaluates** its performance on the validation set.
  - **Prints** the validation accuracy, enabling us to plot and analyze the relationship between training data size and model accuracy.

2.2.3 Model Description

- Our code trains 3 different models from scikit-learn. They are as follows -
  - **Logistic Regression:** Default version of Logistic Regression, which uses and L2 Regularizer and lbfgs solver was used
  - **Support Vector Machine:** SVM with a RBF kernel was used
  - **Multi Layer Perceptron:** An MLP classifier was built using 2 hidden layers, each with 100 nodes each, was used
- Finally the plots are generated based on the accuracies and percentage of training data

2.3 *Text Sequence Dataset*

2.3.1 Data Preprocessing

- We process the input sequences into a format suitable for the neural network.Each sequence in the input str column (assumed to be 50-character strings of digits) is split into individual characters.
- Each character (presumed to be a digit) is converted to an integer using map(int, list(seq)).
- The result is a 2D numpy array where each row is a sequence, and each character in the sequence is an integer.

2.3.2 Training and Validation

- For each percentage (20%, 40%, 60%, 80%, and 100%), the code selects the initial subset of the training data in the respective range. This approach is directly linked to the requirement of Task 1, where we need to evaluate the model's performance on progressively larger subsets of the data.

2.3.3 Model Description

We experimented with various machine learning algorithms and ultimately implemented the classification problem using a hybrid of Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) .This neural network framework was selected to achieve a competitive accuracy on the dataset. The nature of the dataset required leveraging the strengths of CNNs for capturing local patterns in sequences and RNNs for modeling temporal dependencies. This model provide us with several additional layers that enhance our ability to capture the key features in the data such as:

- **Embedding Layer:** Converts integer-encoded sequences into dense vector representations, enabling the model to learn semantic relationships between characters. This helps provide a richer, more meaningful input to the subsequent layers.
- **Conv1D Layer:** Scans the input sequence with a sliding window to capture local patterns and spatial features. It helps identify important character groupings that are relevant for classification.

- **MaxPooling1D Layer:** Reduces the dimensionality of the feature maps, making the model computationally efficient.
- **LSTM Layer:** Captures long-term dependencies and temporal relationships in the sequence. It processes the input in order, preserving context over time, which is critical for sequence-based tasks.
- **Dense Layers:** Fully connected layers that combine the learned features from previous layers to make predictions. The first layer refines the features, while the final layer with a sigmoid activation outputs a probability for binary classification.

The implementation leverages a hybrid neural network architecture combining Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) layers to effectively classify sequences.The LSTM layer captures long-term dependencies within the sequences, allowing the model to understand contextual relationships. Finally, Dense layers refine these learned features to produce a probability output for binary classification. The model is trained on subsets of data to evaluate its performance as more training examples are included, ultimately aiming for optimal accuracy.Our implementation utilizes 10,001 trainable parameters

Task 2:

2.4 *Combining all 3 data sets*

2.4.1 Data Preprocessing

- **Emoticon Dataset(**train_emoticon.csv**):** Each emoji string is separated to 13 columns, each storing a single emoji. One-Hot Encoding is applied on this dataset and the encoder is stored for further use.
- **Feature Dataset(**train_feature.npz**):** The input matrix is flattened to a single dimensional array and PCA is applied on it to reduce the features to 100. The PCA transformer is stored for further use
- **Text Sequence Dataset (**train_text_seq.csv**):** Each string input is separated to 50 columns, each holding a single digit.

2.4.2 Training and Validation

A loop iterates through different percentages of the training data (20%, 40%, etc.), allowing us to observe how the model's performance scales with more data.

For each subset, the code:

- **Applies** the One-Hot Encoding on the emoji subset data, PCA on the features data.
- **Applies** the same transformations on the validation data of emoji and features dataset.
- **Concatenates** the columns of three datasets to create the training set and validation/testing dataset
- **Trains** a classifier using the training data subset.
- **Evaluates** its performance on the validation set.
- **Prints** the validation accuracy, enabling us to plot and analyze the relationship between training data size and model accuracy.

2.4.3 Model Description

- Our code trains 3 different models from scikit-learn. They are as follows -
  - **Logistic Regression:** Default version of Logistic Regression, which uses and L2 Regularizer and lbfgs solver was used
  - **Support Vector Machine:** SVM with linear kernel was used
  - **Multi Layer Perceptron:** An MLP classifier was built using 2 hidden layers, each with 100 nodes each, was used
- Finally the plots are generated based on the accuracies and percentage of training data

3 Results and Observations

3.1 *Emoticons as Features Dataset*

Some of the models and their accuracies on full dataset are given below:

1. Logistic Regression - 89.1
2. KNN - 51.9
3. DT - 76.4
4. SVM - 89.7
5. MLPC - 90.3

Based on these performances, we chose 3 candidate models - Logistic Regression, SVM and MLP Classifier. Detailed accuracies and corresponding plots of these models are given below. MLPC gave the maximum accuracy followed by SVM and Logistic Regression so we selected MLPC as the primary model. On the smallest dataset, all 3 performed roughly the same at around 75% accuracy but on full dataset, only MLPC managed to get the max accuracy of 90+%. Accuracy increases slightly from 60% to 80% which suggests the presence of outliers/less informative data in that section of the dataset. Further more SVM fails the constraint of less than 10,000 parameters so we can't use it as our main model. An ensemble of these 3 candidate models also gave near similar/slightly worse accuracies so we decided to go with an individual model i.e MLPC with 2 hidden layers each containing 100 nodes.

TrainingData%	**MLPC**	**LinearSVM**	**Log. Regression**
20%	76.07%	74.84%	76.28%
40%	80.37%	81.80%	80.57%
60%	84.66%	85.27%	84.46%
80%	87.32%	86.09%	86.91%
100%	91.00%	89.77%	89.16%

Table 1

Figure 1: Logistic Reg.

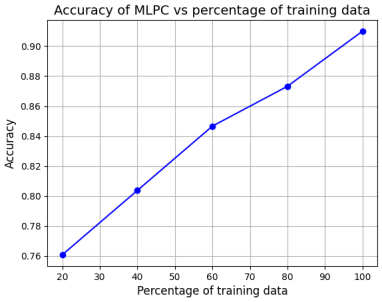

Figure 2: MLPC

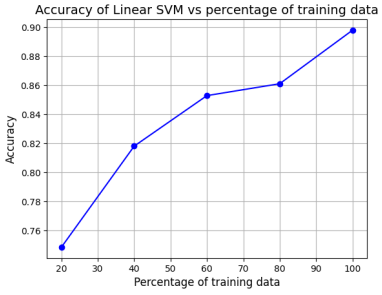

Figure 3: SVM

3.2 *Deep Features Dataset*

Some of the models and their accuracies are given below -

1. LwP - 93.66%
2. Decision Tree - 92.23%
3. Gaussian Naive Bayes - 90.18%
4. SVM poly kernel- 97.75%

TrainingData%	Logistic Reg.	RBF SVM	MLPC
20%	96.32%	96.52%	96.32%
40%	97.34%	96.93%	97.75%
60%	97.95%	97.54%	97.54%
80%	98.36%	98.56%	99.18%
100%	98.56%	98.98%	98.56%

Table 2

Figure 4: Logistic Reg.

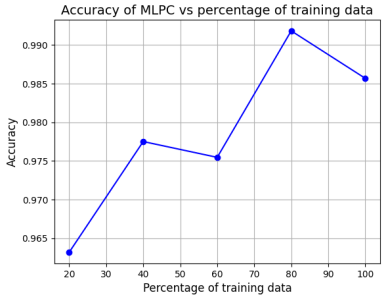

Figure 5: MLPC

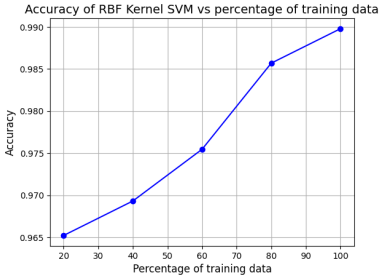

Figure 6: SVM

Similar to Dataset 1, we chose 3 candidate models here - RBF kernel SVM, Logistic Regression and MLPC with 2 hidden layer sizes each with 100 nodes. The accuracies and the plots are shown above. We have noticed that RBF kernel SVM performed consistently well on every percentage of training data followed by Logistic Regression. However, SVM model fails the contraint of ¡10k trainable parameters. MLPC has been a hit or miss in the sense that on every run it either performs very well or very poorly,due to such erratic learning behaviour of MLPC and contraint failure of SVM, Logistic Regression was chosen as our primary model for Dataset 2

3.3 *Text Sequence Dataset*

1. Similarity with Emoticons Dataset : While comparing the raw data of this dataset with the previously used emoticons dataset, we observed a significant similarity between the two. Notably, many emojis corresponded to specific numerical patterns in the strings of our current dataset. However, direct utilization of this insight was challenging due to inconsistent numerical padding across some instances. To handle this complexity, we employed a hybrid neural architecture combining Convolutional Neural Networks (CNNs) with Long Short-Term Memory (LSTM) units. This approach proved to be effective for extracting meaningful features from the sequence data.

2. Performance of Standard Classification Algorithms : Classical machine learning algorithms such as Support Vector Machines (SVM), Logistic Regression, and K-Nearest Neighbors (KNN) performed poorly on this dataset. The primary reason for their failure was the ineffectiveness of these models in dealing with raw data without proper feature extraction. Without robust feature extraction, these algorithms struggled to learn meaningful patterns. However, we hypothesize that these models could achieve better performance if combined with appropriate feature extraction techniques.

3. Ineffectiveness of Kernel-Based Methods : We experimented with kernelizing the input data in an attempt to capture non-linear relationships. However, this approach yielded no substantial improvements. This led us to the conclusion that the dataset required advanced feature extraction methods. Manual feature extraction, although a potential option, would have been labor-intensive and difficult to scale. Therefore, neural network frameworks, particularly CNNs and LSTMs, emerged as the most effective solution for automatically learning features from the input data.

Some of the models and their accuracies on full dataset is given below -

1. kNN(k=7) - 54.98%
2. SVM - 58.53%
3. ANN- 68.71%

4. LSTM - 70.76%
5. CNN-80.16%
6. CNN+LSTM- 85.28%

TrainingData%	**LSTM**	**CNN**	**CNN+LSTM**
20%	63.39%	64.62%	75.26%
40%	67.08%	75.46%	78.12%
60%	68.01%	77.71%	80.57%
80%	66.87%	78.94%	85.07%
100%	70.76%	80.16%	85.28%
Total Trainable Parameters	7,393	7,407	10,001

Table 3

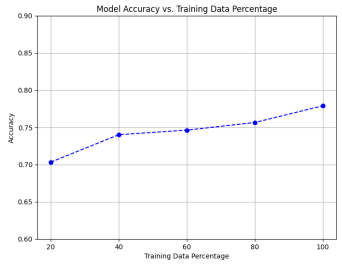

Figure 7: CNN

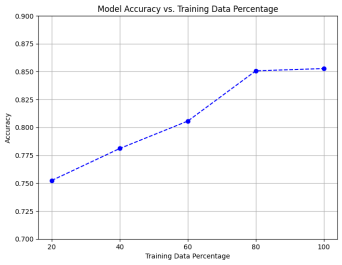

Figure 8: CNN+LSTM

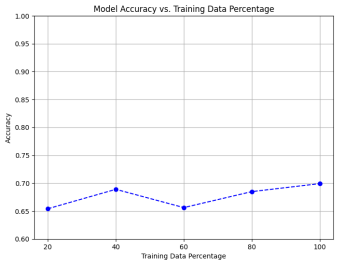

Figure 9: LSTM

Amongst all the models ,CNN+LSTM model provides us the best accuracy overall (85.28%) on the entire dataset and also a very decent accuracy of (75.26%) on the initial 20% of the dataset fitting the criteria of a good classification model that can learn important pattern in the data from a small training set.

3.4 TASK 2 - Combined Dataset

TrainingData%	**Logistic Reg.**	**LinearSVM**	**MLPC**
20%	96.73%	95.70%	89.57%
40%	96.73%	96.52%	92.43%
60%	97.55%	97.14%	93.05%
80%	97.55%	96.93%	93.66%
100%	98.16%	97.75%	94.48%

Table 4

We have only tried 3 models on the combined datasets, those are - Linear Kernel SVM, MLPC and Logistic Regression. The observed accuracies and the plots are given above. We observe the Logistic Regression gave the clear best accuracy compared to the other models. However it is not better than the individual model built for Dataset 2

Figure 10: Logistic Reg.

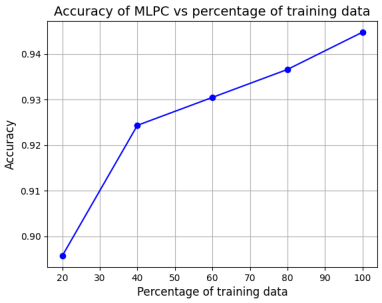

Figure 11: MLPC

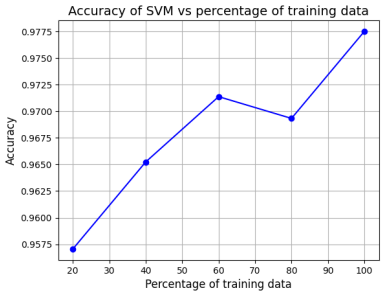

Figure 12: SVM

4 Discussion

The experiments revealed that models like SVM, Logistic Regression and MLPC perform optimally across various feature representations. For instance, MLPC was most effective for the Emoticons dataset, likely due to the categorical nature of the features. Logistics Regression excelled with the Deep Features dataset, benefiting from their ability to handle high-dimensional data. The Text Sequence dataset posed more challenges, with deep learning using the combination of LSTM and CNN providing the best balance between accuracy and training efficiency.

Combining features from all three datasets did not result in better validation accuracy, suggesting that the different feature representations does not provide complementary information and hence reduces the model's predictive capabilities. We have tried using ensemble method both weighted averaging and majority voting both resulting in a subpar result of around 94-95% accuracy hinting at no such strong correlation amongst the dataset points that can be exploited directly to provide a better result.

References

We have primarily used the following libraries:-

1. numpy
2. pandas
3. sklearn
4. tensorflow
5. keras
6. matplotlib

* * *