t-AI-tans Mini-Project 1

Akshat Sharma
230101

Praneel B Satare
230774

Kanak Khandelwal
230520

Dweep Joshipura
230395

1 Task 1a(Emoticon Dataset)

1.1 Data Analysis and Feature Preprocessing

The dataset comprises a total of 214 different emojis, with the frequencies of the top 10 and bottom 10 emojis highlighted for analysis. To prepare this data for modeling, each emoji string in the dataset is converted into a binary matrix with dimensions (214, 13). In this matrix, an entry of 1 at position (i, j) indicates the presence of the i-th emoji in the j-th position of the string.

Subsequently, these binary matrices are flattened into 1D arrays, which serve as input features for the various machine learning models employed in the analysis. This transformation allows for effective handling of the sparse representation of emoji presence in the dataset.

1.2 Models Used and Performance Metrics

The following models were evaluated based on their performance metrics:

Model Architecture	**20%**	**40%**	**60%**	**80%**	**100%**
Perceptron	73.42%	81.60%	84.46%	86.09%	91.21%
Bernoulli Naive Bayes	74.85%	79.96%	83.44%	83.64%	86.09%
Multinomial Naive Bayes	74.64%	81.39%	83.03%	83.84%	86.30%
SVM with Linear Kernel	72.60%	80.37%	83.44%	85.48%	88.34%
Logistic Regression (L2)	73.82%	81.60%	83.23%	87.53%	89.78%
Decision Tree	48.88%	55.42%	71.98%	79.55%	76.28%
Random Forest	66.26%	70.76%	78.94%	83.23%	86.09%
Logistic Regression (L1)	**74.85%**	**82.41%**	**85.48%**	**89.16%**	**93.25%**

Table 1: Model Performance on Various Dataset Splits

Rationale: Each model is chosen based on its suitability for binary classification tasks using sparse binary input data. The Perceptron and Logistic Regression models handle binary outcomes effectively, while Naive Bayes models leverage the independence assumption for sparse features. Random Forest and Decision Trees are used for their robustness and interpretability, and SVM's kernels enhance performance with linearly separable data.

1.3 Logistic Regression (L1)

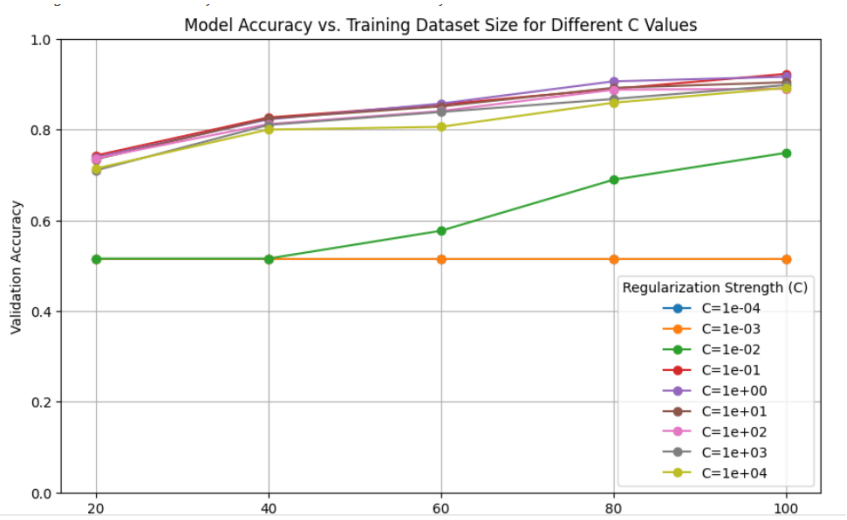
After evaluating the models, Logistic Regression with L1 Regularization was determined to be the best model due to the following reasons:

- It effectively handles sparse input vectors (only 13 ones in a total of 13 * 214 entries).

- L1 regularization automatically selects important features, driving irrelevant feature weights to zero, which is beneficial in high-dimensional sparse data.
- By penalizing large coefficients, it prevents overfitting, leading to improved generalization.

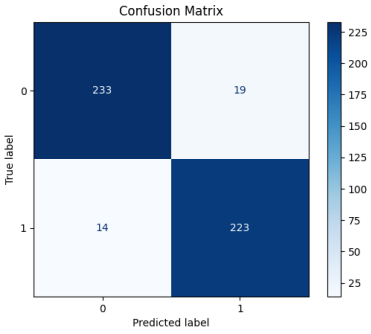
1.3.1 Hyperparameter Tuning

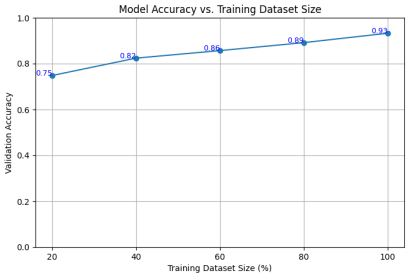
For each fraction of the dataset, a plot of C vs accuracy was generated. The analysis indicated that each model achieved maximum accuracy at C = 0.1. The accuracy increased almost linearly as the fraction of data increased from 20% to 100%. Hyperparameter tuning was conducted using grid search methodology.



1.4 Conclusion

The analysis indicates that the best-performing model was *Logistic Regression with L1 Regularization* due to its ability to handle sparse data effectively and perform feature selection by driving irrelevant feature weights to zero.





2 Task 1b (Deep Features Dataset)

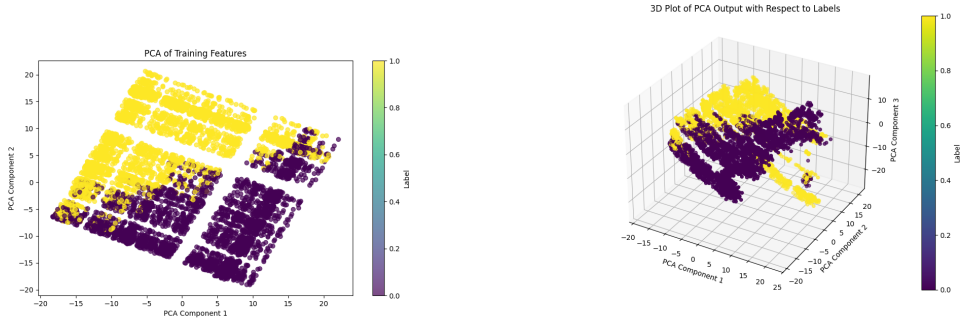
2.1 Analysis of Dataset

The dataset consisted of each input represented by a 13×786 matrix with float entries. To deal with the additional dimension, we attempted two approaches

- Flattening into $10,218$ dimensional vector.
- Principal Component Analysis (PCA) for dimensionality reduction. This was done to bring down the number of features and reducing the training cost and prevent overfitting.

2.1.1 Analysis using PCA

For purposes of visualisation, we applied PCA to the dataset upto 2 dimensions and 3 dimensions.



(a) PCA in two dimensions

(b) PCA in three dimensions

Figure 1: Comparison of PCA in two and three dimensions

We see that the data seems roughly well separated, however there is a significant number of outliers that occur when using such low dimensional representations of the training data.

2.1.2 Analysis using Flattened Vectors

For this analysis, we chose to select 6 data points (three with labels 0, and three with 1). Let the data points with labels 0 be represented by a_i and the latter by b_i. We report the pairwise euclidean distances for the labels below. There does not seem to be significant correlation between the euclidean distances of the flattened training examples and the labels, indicating the absence of a spherical-like separation of the data in the flattened dimension.

	a_1	a_2	a_3	b_1	b_2	b_3
a_1	0.00	0.78	0.90	0.95	0.91	1.00
a_2	0.78	0.00	0.83	0.93	0.90	0.93
a_3	0.90	0.83	0.00	0.93	0.92	0.95
b_1	0.95	0.93	0.93	0.00	0.92	0.97
b_2	0.91	0.90	0.92	0.92	0.00	0.97
b_3	1.00	0.93	0.95	0.97	0.97	0.00

Table 2: Normalised Pairwise Euclidean Distances on Flattened Vectors

For comparison purposes, we also report the pairwise euclidean distances of the same points, but in the 2 dimensional plane given out by transforming the inputs through PCA.

Clearly, there seems to be a lot more correlation after applying PCA (as expected).

2.2 Models implemented

We tried a variety of Classical ML models for this task, each model have been choosed wisely so that it can help provide the best result. Here is a description of why these particular models have been chosen:

	a_1	a_2	a_3	b_4	b_5	b_6
a_1	0.00	0.11	0.13	0.71	0.76	0.56
a_2	0.11	0.00	0.14	0.61	0.68	0.56
a_3	0.13	0.14	0.00	0.65	0.82	0.68
b_1	0.71	0.61	0.65	0.00	0.69	1.00
b_2	0.76	0.68	0.82	0.69	0.00	0.56
b_3	0.56	0.56	0.68	1.00	0.56	0.00

Table 3: Normalised Pairwise Euclidean Distances after PCA (2D) transformations

- **Logistic Regression with L1 or L2 regularization:** Suitable for binary classification tasks. Regularization (especially L1) helps reduce the risk of overfitting by penalizing large coefficients, useful when dealing with many features.
- **Support Vector Machines (SVM):** Particularly effective for high-dimensional spaces. Kernel trick allows it to model complex decision boundaries.
- **Random Forest:** Ensemble methods like Random Forest handle high-dimensional data well and can model non-linear relationships. Feature importance can also help you understand which features are most relevant.
- **k-Nearest Neighbors (k-NN):** While computationally heavy in high-dimensional spaces, it can work well if combined with dimensionality reduction techniques like PCA.

Validation set accuracy for every dataset split is given in the table below. This analysis was done to identify the best overall model architecture, after which we worked on hyperparameter tuning.

Model Architecture	**20%**	**40%**	**60%**	**80%**	**100%**
Random Forest Classifier	0.8814	0.9243	0.9202	0.9305	0.9223
Decision Tree Classifier	0.9059	0.9284	0.9468	0.9591	0.9407
K-Nearest Neighbors (DTW distance-metric)	0.9100	0.9121	0.9059	0.9080	0.9162
Logistic Regression (L1 Regulariser)	0.9632	0.9816	0.9775	0.9836	0.9816
Logistic Regression (L2 Regulariser)	0.9427	0.9714	0.9755	0.9836	0.9816
SVM (Linear Kernel)	0.9509	0.9632	0.9775	0.9816	0.9796
SVM (Polynomial Kernel)	0.9632	0.9714	0.9755	0.9836	0.9877
SVM (RBF Kernel)	0.9509	0.9755	0.9796	**0.9918**	0.9877

Table 4: Model Performance Across Different Dataset Splits

2.3 Preprocessing

In addition to using PCA for several of our implementations, we performed Standard Scaling, which involves getting mean $\mu = 0$ and standard deviation $\sigma = 1$. This lead to an accuracy increase of around 3% on the validation set, when tested with the highest accuracy model. The above reported accuracies are post-scaling.

2.4 Tuning SVM

In addition to testing with three kernels (linear, polynomial and rbf), we tuned the hyperparameter C for the SVM, by evaluating validation accuracy for different values of C.

2.4.1 C hyperparameter of SVM

The C parameter in SVM allows us to control the trade-off between the margin and misclassifications. A smaller value of C promotes a wider margin but allows more misclassifications, while a larger value of C prioritizes correct classification at the expense of a narrower margin.

2.5 Results of SVM

The confusion matrix and validation accuracy with dataset split are plotted below.

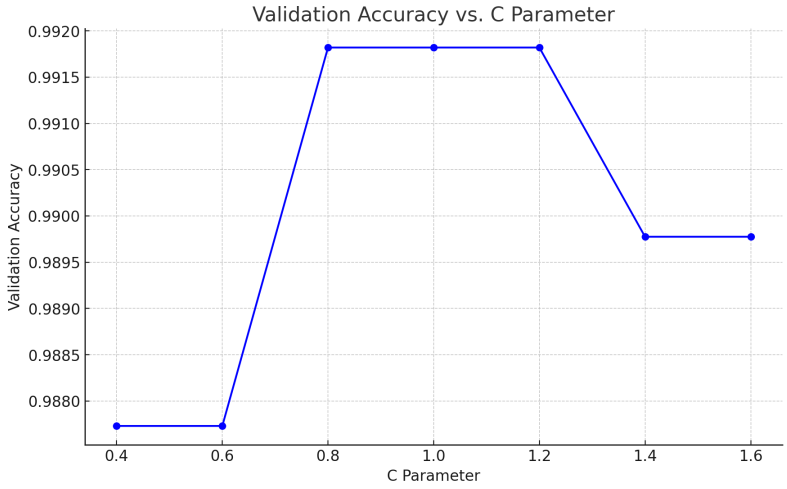
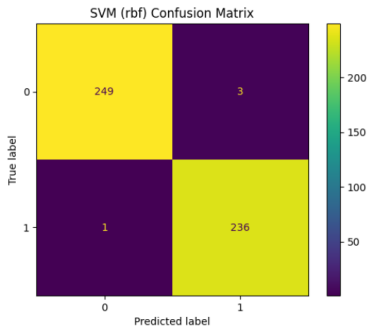
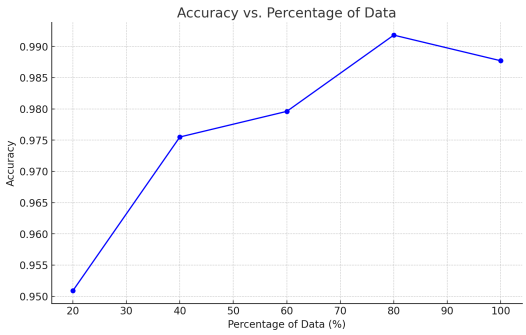


Figure 2: Results after tuning C



(a) Confusion Matrix for SVM



(b) Training data vs Validation Accuracy

Figure 3: Analysis of SVM Results

3 Task 1c

Model Architecture	**20%**	**40%**	**60%**	**80%**	**100%**
Logistic Regression	0.5706	0.5399	0.5133	0.5112	0.5378
Random Forest Classifier	0.5849	0.6115	0.6074	0.6053	0.6380
K-Nearest Neighbors	0.5215	0.5297	0.5419	0.5031	0.5358
XGBoost	0.5767	0.5746	0.5726	0.5971	0.5930
SVM (Linear Kernel)	0.5521	0.5337	0.5235	0.5235	0.5256
SVM (Polynomial Kernel)	0.5378	0.5440	0.5297	0.5256	0.4949
SVM (RBF Kernel)	0.5746	0.5787	0.5890	0.5746	0.5583
Multilayer Perceptron	0.5174	0.4949	0.5399	0.5112	0.5439
Fully Convolutional Network	0.5256	0.5603	0.5174	0.6912	0.6155
LSTM	**0.5971**	**0.6319**	**0.6830**	**0.6912**	**0.7587**

Table 5: Model Performance on Various Dataset Splits

The data is provided in the form of a string of digits, of length 50, for each example. For training and evaluation of the various models, the training and validation data (comprising 7080 and 489 samples each) were stored in arrays of shape (number of examples, 50).

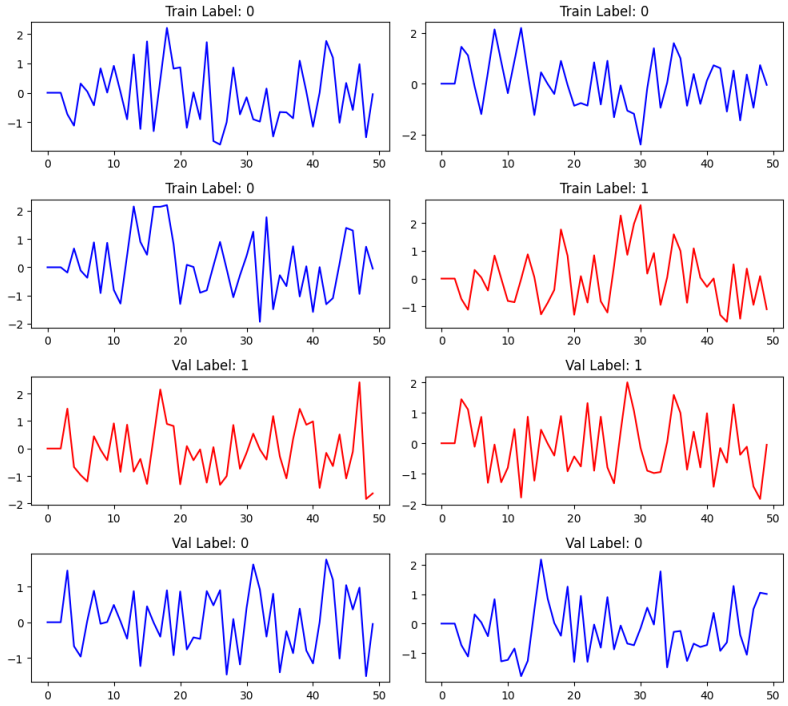
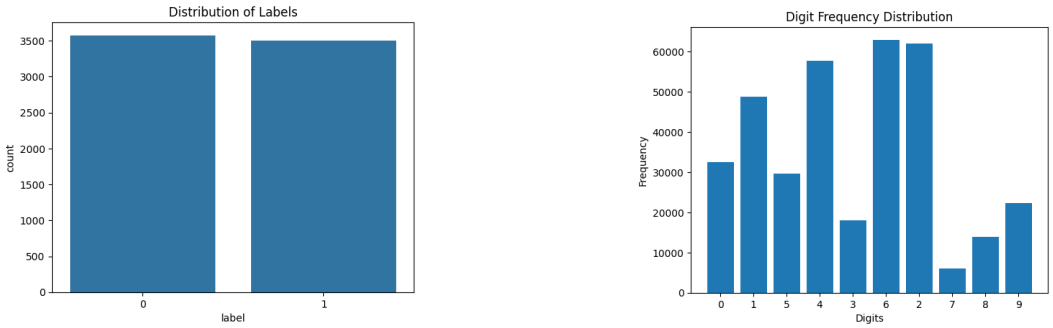


Figure 4: Standardized data

3.1 Data analysis

The distribution of labels among the training examples indicates a balanced dataset with data evenly distributed between the binary labels, 0 and 1. Next, the digit frequency distribution is plotted. The



(a) Distribution of labels

(b) Digit frequency distribution

Figure 5: Analysis of the distribution

training and validation text sequences consisting of digits (0-9) are standardized for better accuracy and efficiency. This is done by computing the mean μ and standard deviation σ for each column separately, and applying the following transformation on each digit in the respective column:

$$x^{'} = \frac{x - \mu}{\sigma}$$

The standardized data has mean 0 and variance 1 along each column (each feature, i.e. for a particular position in the sequence). We plot a few examples (after standardization) from both, training and validation datasets, as feature values in sequential order (elements of the sequence).

3.2 Classical models

Various classical machine learning models including Logistic Regression, K-Nearest Neighbors and Decision Trees are trained on different amounts of the standardized data - 20, 40, 60, 80 and 100 percent. Training is followed by evaluation on the validation dataset in terms of accuracy, precision, f1-score and confusion matrix. Performance on the validation data is poor with most models giving below 60 percent accuracy. The random forest classifier yields maximum accuracy of 63.8 percent on 100 percent training data.

The flat performance of linear models (Logistic Regression and Linear SVM) suggests that the binary classification problem cannot be solved by simple linear combinations of the digits. The relationships in data are likely nonlinear. The Random Forest and XGBoost models, which are known for capturing complex interactions between features, also perform well as the dataset size increases. This suggests that the relationships between digits at different positions in the sequences may not be linear, and more advanced models are needed to exploit these interactions.

3.3 Neural Networks

We trained three neural networks on the data - a multilayer perceptron, a fully convolutional network and an LSTM (Long short term memory) network. This was followed by evaluation on the validation data. The somewhat better performance of the fully convolutional network indicates that local digit patterns (e.g., specific digit groups) may be significant for classifying the sequences.

While the used classical models and feedforward neural networks are adept at learning patterns in data, they are unable to capture the sequential characteristics of data, i.e. data is processed only in the forward direction and the concept of memory does not arise. In the text sequences dataset provided, the categorization of samples appears to be linked to the sequential features. In other words, it might be a time series dataset. This explains why classical models and feedforward networks perform poorly during evaluation on validation set.

Recurrent Neural Networks have been developed for capturing sequential information. These are "are a class of artificial neural networks for sequential data processing. Unlike feedforward neural networks, which process data in a single pass, RNNs process data across multiple time steps, making them well-adapted for modelling and processing text, speech, and time series."

LSTM is a type of Recurrent Neural Network (RNN) that can detain long-term dependencies in sequential data. Appreciable results are achieved with a neural network comprising one LSTM layer with 32 units and one dense layer of a single neuron with sigmoid activation (for obtaining probabilities). This model yields the maximum accuracy of 76.48%.

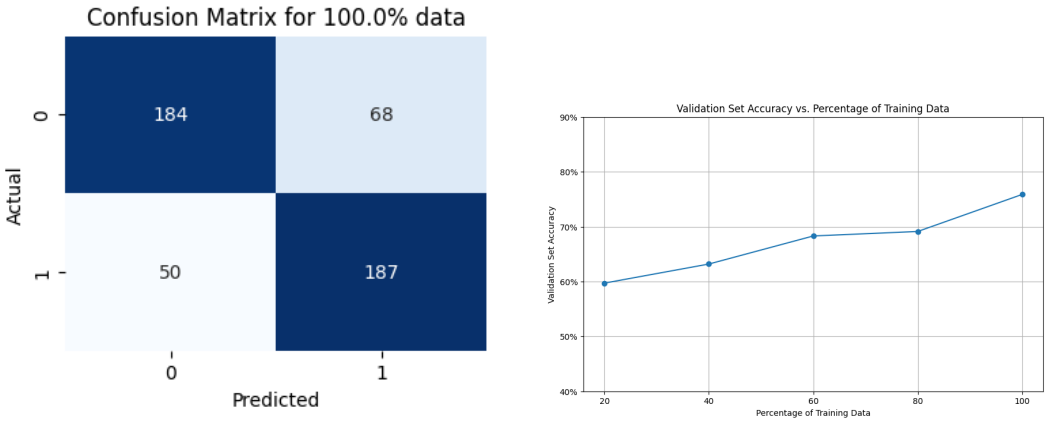
The digit sequences have strong sequential dependencies, where the order of digits plays a critical role in determining the label. LSTM's strong performance highlights this.

Hyperparameter tuning comprising of learning rate and batch size optimization is done using the evaluation metrics, primarily accuracy. For the LSTM, following are the results with learning rates of 0.001, 0.005, 0.010, 0.050 and 0.100, each with batch size 128, after training for 75 epochs.

Learning Rate	**Maximum accuracy(% of training data)**
0.001	0.63 (80%)
0.005	0.74 (100%)
0.010	0.76 (100%)
0.050	0.53 (40%)
0.100	0.52 (60%)

Table 6: Model performance after learning with different learning rates for 75 epochs (Adam optimizer with batch size 128)

Evidently, a learning rate of 0.01 is optimal. With this learning rate, batch sizes of 128, 64, 32 and 16 are taken and validation accuracy compared.



(a) Enter Caption for Confusion Matrix

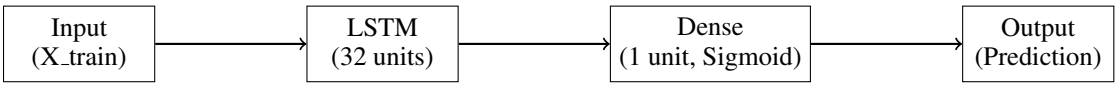
(b) Enter Caption for Accuracy vs Percentage

Figure 6: Combined Figure: Confusion Matrix and Accuracy vs Percentage

Batch Size	**Maximum accuracy(% of training data)**
256	0.75 (100%)
128	0.76 (100%)
64	0.73 (100%)
32	0.74 (80%)
16	0.72 (100%)

Table 7: Model performance after learning with different batch sizes for 75 epochs (Adam optimizer with learning rate 0.01)

Thus we obtain the following final best model trained on 100% of the training data, using the Adam optimizer with a learning rate of 0.01 and batch size 128. This model yields 75.87% validation accuracy.



Layer	**Output Shape**	**No. of parameters**
LSTM	(None, 32)	4,352
Dense	(None, 1)	33
	Total params:	4,385 (17.13 KB)
	Trainable params:	4,385 (17.13 KB)
	Non-trainable params:	0 (0.00 B)

Table 8: Best model for learning text sequences dataset

For this model, the confusion matrix as well as the plot of validation accuracy achieved against % of training data used, is shown in figure 6.

3.4 Conclusion

In this task, we analyzed various classical machine learning models and neural networks to classify digit sequences. The findings revealed that while classical models struggled with the non-linear relationships present in the data, the LSTM architecture achieved superior performance by effectively capturing the sequential dependencies within the digit sequences. The experiments conducted on hyperparameters underscored the significance of careful tuning to achieve optimal model performance.

4 Task 2

4.1 Introduction

In this task, we had to use the combined dataset to yield outputs, that would give the maximum accuracy. We considered approaches that could use all the data simultaneously. We settled on using a weighted approach, where the outputs of the different models are used according to their efficacy on the validation set.

Model (Task)	**Maximum Validation Accuracy (%)**
Logistic Regression (1a)	0.933
SVM with RBF Kernel (1b)	0.992
LSTM (1c)	0.808

Table 9: Accuracies of best models, trained on different tasks

4.2 Problems

A major obstacle while designing a model (or an ensemble of models) that is designed to increase accuracy, is that the SVM model already has a validation accuracy of over 99%, which is significantly higher than that of the other two models. Hence, there isn't sufficient scope for improvement, when using a combination of datasets, in terms of an increase in validation accuracy. If we were to have another "harder" labelled set of inputs, it would be sensible to train a separate classifier for the combination of datasets.

On the other hand, using a variety of models, trained on separate features, gives us lesser susceptibility towards noise. With this rationale, we attempted approaches to use an ensemble that uses the outputs of the pre-existing (best) models, and uses that to yield a good accuracy on the validation set.

4.3 Ensemble Learning

The three major approaches, listed in increasing order of computational complexity are

1. **Majority Voting**: In this approach, every model gets a vote, and the final output depends on the results.
$$\text{Class} = \mathbb{I}(\sum_{i=1}^{3} y_i \geq 2)$$
2. **Average Voting**: In this, the final class is chosen by averaging the outputs of the different models.
$$\text{Class} = \mathbb{I}\left(\frac{\sum_{i=1}^{3} y_i}{3} \geq 0.5\right)$$
3. **Weighted Voting**: Every model is given (trained or tuned) weights, and the final result is determined on the weighted average of the outputs of individual models.
$$\text{Class} = \mathbb{I}\left(\frac{\sum_{i=1}^{3} \alpha_i y_i}{\sum \alpha_i} \geq 0.5\right)$$

In the case of binary classification using 3 models, we see that Majority Voting and Average Voting would yield the same outputs (as two models would dominate the output of the other model).

4.3.1 Learning Weights for Weighted Voting

For weighted voting, the α_i were considered parameters, and SLSQP (Sequential Least SQuares Programming) was used (as an import from `scipy` using `scipy.minimize()`).
The optimization function was

$$\boldsymbol{\alpha} = \arg\min \frac{\sum_{i}^{N} |\hat{y_i} - y_i(\boldsymbol{\alpha})|}{N}$$

where

$$\hat{y_i} = \mathbb{I}\left(\frac{\sum_{i=1}^{3} \alpha_i y_i}{\sum \alpha_i} \geq 0.5\right)$$

4.4 Results

Method	Validation Accuracy (%)
Majority Voting	0.953
Learned Weighted Voting	0.992

Table 10: Accuracies of Ensemble Models

When analysing the weights given by the optimizer, we found that the α corresponding to SVM trained on task (2) was higher than the sum of the α's of the other two tasks, yielding the same outputs as the SVM. To be specific, it gave the weights as $[0.15, 0.7, 0.15]$. Hence, while it could give a possibly more reliable confidence score, the accuracy and the final prediction would remain the same, when using the ensemble or the SVM.

4.5 Limitations

While performing this task, it was evident to us that ensemble methods were most useful when we had probabilities/confidence scores as the outputs from the models, which should then be used instead of output predictions. However, as SVMs do not output confidence scores, this was infeasible for our current selection of models.[1]To alleviate this, we can use probabilistic models for all three tasks, outputs of which can be combined in a more natural manner when designing and evaluating ensembles, in a future improvement.

4.6 Conclusion

As per the above discussion, we used the SVM trained and tuned in task 2, as our final model, for the combined dataset. This approach yielded the validation accuracy equivalent to our Task 1b, i.e., 99.2%.

[1]The closest analog to a confidence score can be the (inverse of) distance from the decision boundary. However, these do not directly correspond to probabilities, and were hence not used.