CS771 Mini-Project 1

R.Charan
230819

Vedhanth Balasubramanian
231135

Aritra Ray
230193

Sanjna S
230918

Ashwin Barnwal
230234

Contents

1 Introduction **1**

2 ML Models **2**

3 Task 1 **2**

3.1 Dataset I: Emoticons 2

3.2 Dataset II : Deep Features 3

3.3 Dataset III: Text Sequences 7

4 Task 2 **9**

4.1 Approach 1: FeaturesDataset 9

4.2 Approach 2: LogitsDATASET 10

4.3 Results 11

5 Conclusion **11**

1 Introduction

In this project, we aim to identify the best-performing binary classification models for two distinct tasks using three different datasets. These datasets, generated from the same raw data but represented with varying feature sets, allowed us to experiment with multiple machine learning models and feature engineering techniques. Task 1 focuses on finding the most efficient and accurate model for each individual dataset by training and evaluating several models based on accuracy and training data size. Task 2 extends this analysis by combining all three datasets to investigate whether a unified model can outperform the individual models.

For each task, models were trained using varying portions of the training data, and the validation set accuracy was used to select the best models. We also experimented with different feature transformations to improve performance, particularly for the complex dataset representations. Ultimately, the best models were chosen based on their ability to balance high validation accuracy with minimal training data usage, ensuring both cost efficiency and generalization capability.

2 ML Models

The following common ML models have been used in each dataset. Thus we have provided here a common summary of the models and how they have been implemented.

- **Support Vector Machine (SVM):** We use *sklearn.svm* to train a soft-margin SVM with *rbf* kernel transformation as on the training data set. This is a good model to try since it is very reliable since a good margin is created. We have done hyper-parameter tuning of C for SVM loss using *GridSearchCV* with 'C': [0.001, 0.01, 0.1, 1, 10, 100].
- **Logistic Regression:** Given that the task is of binary Classification, Logistic Regressions seemed fit to try. Using the *LogisticRegression* function of *sklearn.linear_model* library we implemented our model.
- **Neural Network:** We tried Neural networks since they are good at learning complex data. We implemented a simple neural network using *Tensorflow.keras* library. The layers are chosen appropriately keeping in mind the 10000 trainable parameter limit. We trained the model using *BinaryCrossEntropy loss function*, and *Adam Optimizer* with 0.001 learning rate.
- **Random Forest:** We have used *RandomForestClassifier* from *sklearn.ensemble*, followed by hyperparameter tuning using Optuna.
- **XGBoost:** We utilise XGBoost using *XGBClassifier* from the xgboost library, followed by hyperparameter tuning using Optuna.

3 Task 1

3.1 Dataset I: Emoticons

The features of each input are given in form of 13 emoticon. We first split it into thirteen columns with each containing one emoji.

3.1.1 Data Pre-Processing

An initial approach we considered was to process the emoticons as their unicode values, thus treating them as numbers and working from there. However upon training a neural network on the data we get poor performance. This is because there is no sense of linearity or comparison between the unicode values. An emoji having a smaller unicode value than another is meaningless for our purposes.

Thus we used **one hot encoding** instead, converting it into sparse inputs. For this we used the *OneHotEncoder function* from *sklearn.preprocessing* library.

This modified our input to have 2159 columns. This may be a large number but it's still easy to process since they are sparse vectors. Moreover this takes into consideration both the **unique emoji** and **its position**.

3.1.2 Methodology

The processed input data was then modified using functions from the optuna li used to train multiple ML models such as Logistic Regression, Support Vector Machine (SVM) and Decision Trees (DTs) as mentioned under the **ML Models** section. In each case, the model was trained on the first 20%, 40%, 60%, 80% and 100% of the training data. The models all were then made to predict the labels for the validation set and accuracies were calculated in each case. The following hyperparameters were used respectively.

SVMs: C=3.010902863513705, kernel='linear'

RF: n_estimators=250, max_depth=500, min_samples_split=4, min_samples_leaf=2, max_features=0.8, bootstrap=True

XGB: n_estimators=500, max_depth=36, learning_rate=0.33, subsample=1.0, colsample_bytree=0.1, gamma=0.3486367352523927

3.1.3 Plotting the results

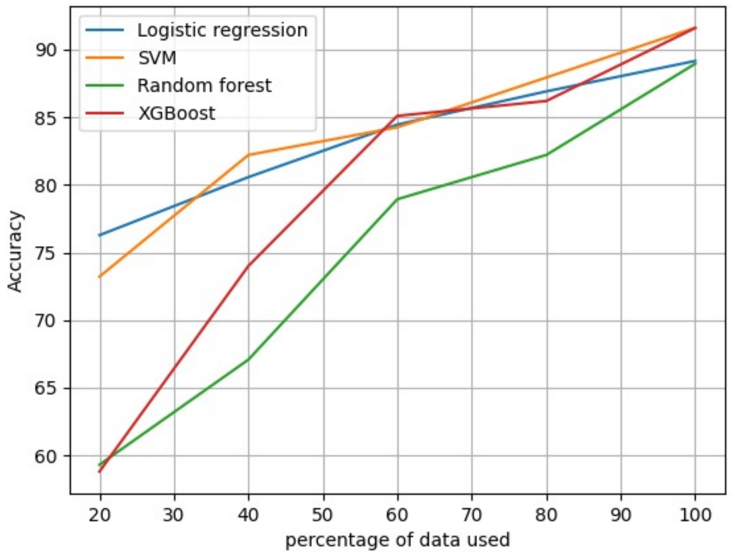


Figure 1: Emoticon Dataset

3.1.4 Results

The best model for this dataset is **SVMs**. As we can clearly see that it has the highest validation accuracy and it also generalises well for smaller datasets.

The validation accuracies for the used svm model for the above mentioned percentages of training data are **[73.21%, 82.21%, 84.25%, 87.93%, 91.61%]** respectively.

3.2 Dataset II : Deep Features

In this dataset, we were provided with 13×768 matrices for each of the inputs, which were basically 768 dimensional embeddings of each of the 13 emoticons, extracted using a neural network.

The challenge in working with this dataset was the sheer number of features and hence the computational complexity, for example if we linearize all the matrices into vectors of size (13*768 = 9984) and train even a simple neural network on it, the number of trainable parameters exceeds 10000. Thus we had to find a clever way to extract relevant features from the matrices and reduce the dimensions to something more manageable.

3.2.1 Exploring the dataset

First of all we conducted a basic exploration of our training dataset. To explore each 13×768 array, we **linearized** it to vectors of 13*768 = 9984 size. Then we applied standardization (also called **Z-score normalization**) on each input of the dataset, which transforms each value of the input array using the formula: $X_{new} = \frac{(X - mean)}{std}$.

On standardizing, we found that 7680 of the values in each input vector had become zero! Hence we had just (9984 – 7680) = 2304 non-zero entries in each vector. This further suggested that the given matrices of size 13 × 768 had many redundant values. Hence we proceeded with several feature extraction and dimensionality reduction approaches as outlined below.

3.2.2 Approach 1: Averaging the 13 × 768 matrices to 768 dimesional embeddings

To start off we tried the most obvious and naïve approach of averaging all the input matrices to vectors of size 768.

Rationale : This will reduce the dimensions of the input matrices, making it easier to handle, and each of the 768 dimensional embeddings will roughly represent the respective inputs. We also standardized (Z-score normalization) the input vectors as ML models work better with standardized data. Next we tried to fit a simple neural network and a decision tree model.

We obtained a peak accuracy of just 54% (for the neural network). Thus this approach was rejected, and we have tried other approaches for pre-processing the data. This poor performance of averaging might be because averaging eliminates the sequential or positional information, which might be crucial for the classification.

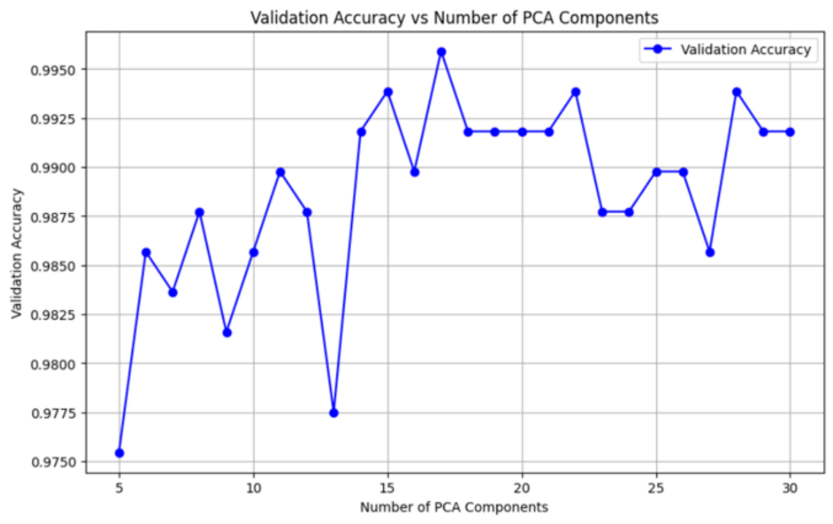
3.2.3 Approach 2 : Principal Component Analysis (PCA)

We tried Principal Component Analysis (PCA) next, a very useful unsupervised learning algorithm for dimensionality reduction.

Rationale : PCA gives a very good representation of high dimensional data, by giving us orthogonal axes which are linear combinations of the original variables and which capture the maximum variance in our data. Thus PCA reduces our data to very low dimensions without much loss in data.

We used PCA algorithm from Sklearn.decomposition library. The pipeline we followed –

i) We treated n (no. of dimensions PCA will reduce our data to) as a hyperparameter. We tried values of n from 5 to 30, fitting a simple baseline neural network consisting of 4 layers and just 753 trainable parameters on the whole training dataset, and measuring the validation accuracies. The results from varying the value of n has been plotted in the graph below.



```
Best number of PCA components: 17 with validation accuracy: 0.9959
```

Figure 2: Validation Accuracy vs Number of PCA components

We chose **n=17** which gave the highest validation accuracy of **99.59 %** (note this was after using the whole training dataset).

ii) Next we chose the first 20%, 40%, 60%, 80% & 100% of the data and applied PCA with 17 dimensions on it. Then we tried 3 models in each case – a) a deep neural network, b) Random Forest Classifier, c) XGBoost Classifier, as explained in the **ML Models** section.

Deep Neural Network: For each of the fractions of the training dataset, we fit a 4 layer densely connected neural network with 170, 34, 17 and 1 neurons respectively in each layer. All the neurons of the first 3 layers had 'ReLU' activation function while the last neuron in the last layer had 'sigmoid' activation function. We used BinaryCrossEntropy loss function and Adam Optimizer with learning rate 0.001 for training. The total number of trainable parameters of the model was 9,487. (Note : The number of layers, number of neurons in each layer, learning rate, etc. was chosen after some manual finetuning.)

Random Forest: We found the following hyperparameters to be the best.

i) n_estimators: 179

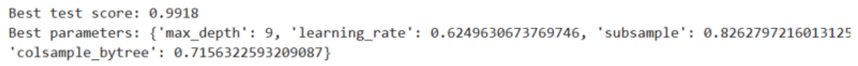
ii) min_samples_split: 8

iii) min_samples_leaf: 2

iv) max_features: log2

v) criterion: entropy

XGBoost: The best hyperparameters found were –



```
Best test score: 0.9918
Best parameters: {'max_depth': 9, 'learning_rate': 0.6249630673769746, 'subsample': 0.8262797216013125
'colsample_bytree': 0.7156322593209087}
```

Figure 3: The best hyperparameters for the xgboost model

Plotting the results:

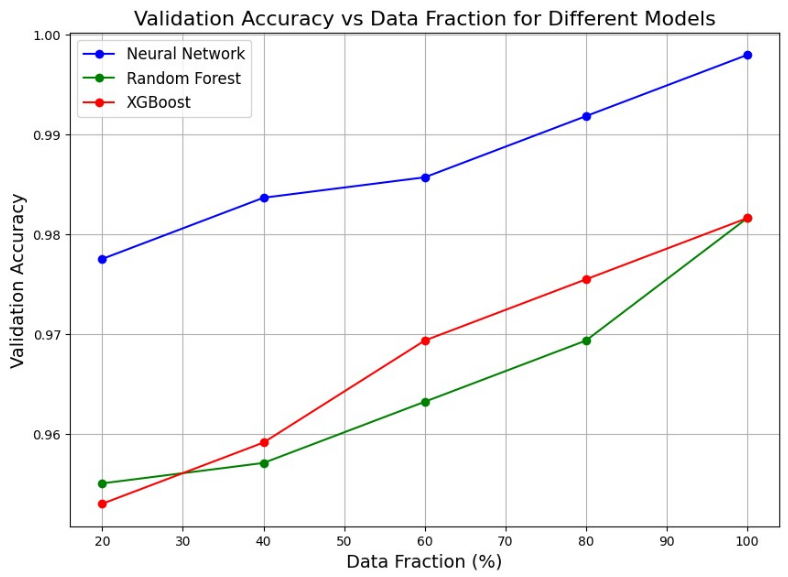


Figure 4: Principal Component Analysis

The best model in this approach is the neural network, with accuracies **97.75 %**, **98.36 %**, **99.18 %**, **99.18 %** and **99.59 %** respectively.

3.2.4 Approach 3: Linear Discriminant Analysis (LDA)

To the flattened 13×768 matrix, we apply standardisation and remove redundant features. To the remaining 2304 feature dataset, we apply LDA. **Rationale:** LDA is very promising since unlike other dimensionality reduction methods like PCA which are unsupervised, LDA is supervised and considers the labels while performing the dimensionality reduction. The number of trainable parameters too is well within the limit.

The dimensions have been directly **reduced to 1** since LDA reduces dimensions to ATMOST $(no.of classes - 1)$. LDA has been implemented using *sklearn.discriminant_analysis*.

Now the transformed data has been used to train various models as shown in the **ML Models** section. Apart from these models, LDA itself can be directly used to predict the label. Using LDA directly is simpler and provides a straightforward classification model. LDA alone assumes linear separability, in contrast to using LDA for dimensionality reduction followed by another classifier allows for capturing non-linear relationships in the data, as long as the subsequent model is capable of doing so. (Like Decision tree based models & SVM with kernel.)

The models each are trained for the first 20%, 40%, 60%, 80% and 100% of the training data and the accuracies of each on the validation set have been plotted.

Random Forest: Using Optuna, best-parameters: 'n_estimators': 163, 'max_depth': 10, 'min_samples_split': 2, 'criterion': 'gini', 'min_samples_leaf': 4

XGBoost: Using optuna, best-parameters: 'n_estimators': 181, 'max_depth': 8, 'learning_rate': 0.29531266150659247, 'subsample': 0.7412925230234721, 'colsample_bytree': 0.6019104807323039

Soft-Margin SVM: Using GridSearchCV, best-parameter C: 0.1, kernel: 'rbf'

Plotting The Results:

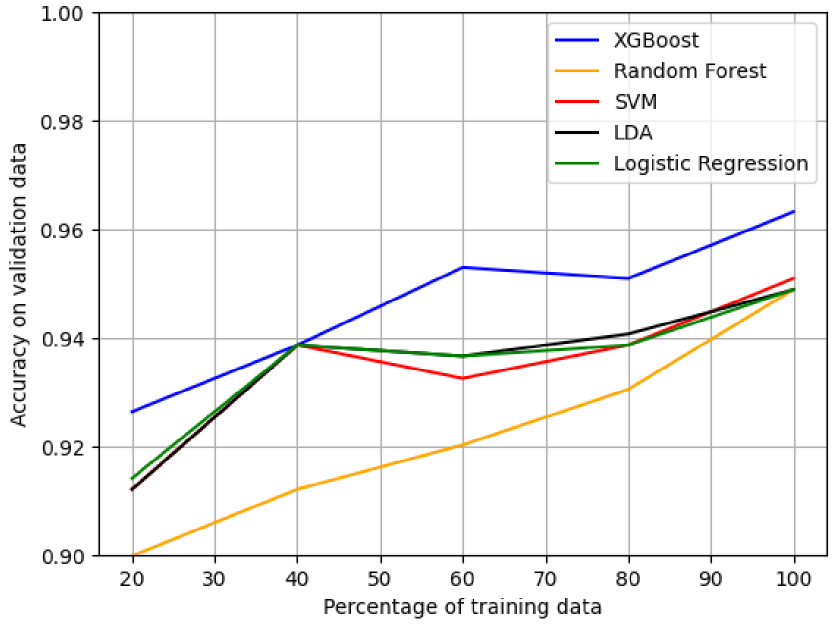


Figure 5: Linear Discriminant Analysis

*Among the models tested, the best one is **XGBoost** with the following accuracies on each percentage of training data,*

[92.63%, 93.86%, 95.29%, 95.09%, 96.32%]

A reason why XGBoost worked best could be because XGBoost is a non-linear model which was applied on top of the linear dimensionality reduction that LDA did.

3.2.5 Approach 4: Auto-Encoder

We tried auto-encoder next, a very useful method for dimensionality reduction. The number of features that we reduce it to depends on the number of trainable parameters and the accuracy it provides on the model. Even though reduction to 256 features was the best, due to the number of trainable parameters limit, we are reducing it to just **1 feature**. (6913 trainable parameters). This has been done using Dense and layers from *tensorflow.keras*. The trainable parameters have been ensured to be below the limit of 10000 by checking with *autoencoder.summary()*.

Models in this approach could only reach a maximum accuracy of around 82% that too on 100% of the data, and thus we can see that the auto-encoder approach isn't a good one for this data set as compared to PCA or LDA.

3.2.6 Results

Both PCA and LDA based approaches were good. The best one however was from **PCA based dimensionality reduction, followed by Deep Neural Network** to train the transformed training set. This was chosen as the best model since it has the highest accuracy for each percentage of training data.

A reason as to why Deep Neural Network gave the best result could be because it can learn complex associations from the given data and it is especially suited for features with numerical values (array-type numerical data), as is the case here.

3.3 Dataset III: Text Sequences

This dataset consisted of digit sequences of length 50 where each character ranges from 0-9.

3.3.1 Feature Extraction

We tried various methods of feature extraction, some are listed below:

- **N - grams:** Broke the input string into substrings of length n, where n ranged from 1 - 50. Then used the value of each substring as a feature.
- **Count of substrings:** Realising there might be information extracted from the sequence/order of digits, we first pre calculated all the possible substrings of length < 10 (which can be made by digits) then used the count of substrings in the input as individual features.
- **One hot encoding:** We treated each index of the input as an individual categorical feature with 10 possible outputs and performed one hot encoding giving us $50 * 10$ columns.

We trained **ML Models** on these features such as logistic regression and Random Forest. We got very less (about < 50) accuracies for the first two features. Moreover, the computation costs for the first two features were also relatively much higher. So, we decieded to train and test any further models with the third method of feature extraction where we had got > 60.

3.3.2 Dimensionality Reduction: Principal Component Analysis (PCA)

After the feature extraction, we have 500 features. Since basic ML Models like **Logistic Regression, Random Forests and Support Vector Machines(SVMs)** would work bettter with lesser dimensions, we apply dimensionality reduction using Principal Component Analysis.

We used the number of principal components n as a hyper-parameter and choose that n which gave the best accuracy on the vaidation dataset. To accomplish this, for each model we plotted the validation accuracy for each n and found out the required n. The results are tabulated below:

ML Model	n - Hyperparameter	Validation Accuracy
Logistic Regression	392	68.5%
Random Forest	430	63%
SUpport Vector Machines	395	68%

Table 1: ML Model Performance with Hyperparameter n

We got descent accuracies with these models and PCA didn't seem to work out as much as expected since as the best results still come out with very high number 400 components. So, to handle larger dimensions more efficiently and try a more complex ML Models, we tried a deep learning approach using neural networks.

3.3.3 Convolutional Neural Networks(CNN)

We thought CNN would be a great fit for this task because of the following reasons:

- **Handle Higher Dimensions:** A neural network can train itself and find out the most important features and hence handle higher dimensions efficiently.
- **Local Pattern Recognition:** CNNs can learn patterns in sequences, like recurring digits or clusters.
- **Hierarchical Feature Extraction:** Pooling layers allow CNNs to learn both fine-grained and high-level patterns.

3.3.4 Model Architecture

The CNN model used the following architecture:

- **Input Layer:** Input shape of $(50, 10)$ representing 50 digits with 10 possible categories (0-9).
- **Conv1D Layer 1:** 32 filters, kernel size of 3, activation function ReLU.
- **MaxPooling1D Layer 1:** Pool size of 2.
- **Conv1D Layer 2:** 64 filters, kernel size of 3, activation function ReLU.
- **MaxPooling1D Layer 2:** Pool size of 2.
- **Dropout Layer:** Dropout rate of 40% to reduce overfitting.
- **Flatten Layer:** Converts the pooled feature maps into a 1D vector.
- **Dense Output Layer:** Single neuron with sigmoid activation for binary classification.

The CNN was compiled with **Adam optimizer**, a learning rate of 0.005, and **binary crossentropy loss**. The model was trained for 100 epochs with a batch size of 10.

3.3.5 Results

The CNN achieved the following performance metrics:

- **Training Accuracy:** 96.81%
- **Validation Accuracy:** 93.25%

The CNN model outperformed other classical models, demonstrating the effectiveness of deep learning for sequence-based classification.

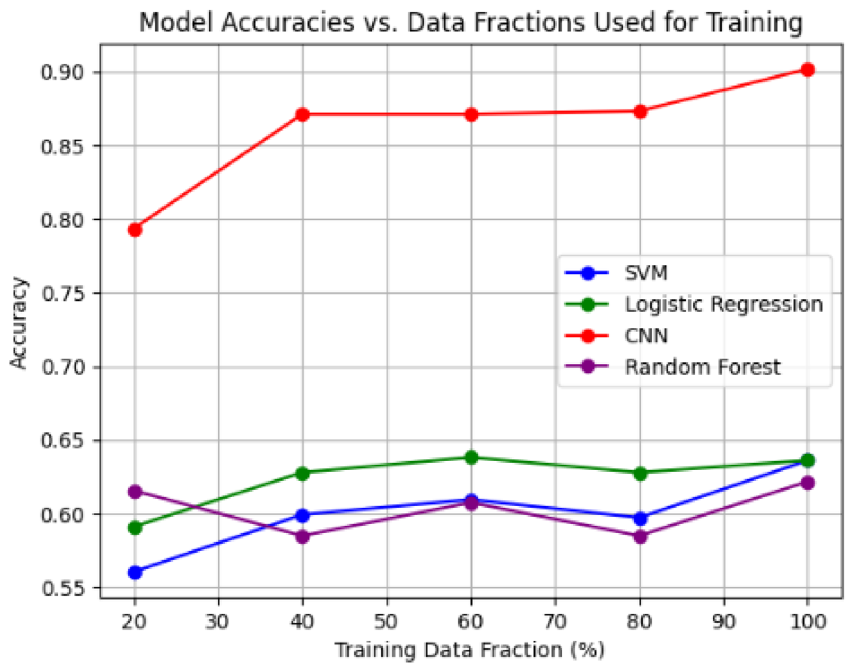


Figure 6: Test Accuracy vs Percentage of Training Data Used.

Accuracies of the ML Models used:

Logistic Regression: $[59.1, 62.7, 63.8, 62.7, \mathbf{64.5}]$

Random Forest: $[61.5, 58.4, 60.7, 58.4, \mathbf{63}]$

SVM: $[56, 60, 60.1, 59.7, \mathbf{63.9}]$

CNN: $[79.3, 87.1, 87.1, 87.3, \mathbf{93.2}]$

The graph above shows the relationship between the percentage of training data used and the test accuracy achieved for the different models used for this task and dataset. As expected, increasing the amount of training data improved the performance of the model.

The CNN model achieved the highest accuracy (93.25%) among all the models we tested. This highlights the strength of deep learning methods, particularly for tasks that involve sequential patterns.

4 Task 2

We have tried out combining the datasets in 2 ways - FeaturesDATASET and LogitsDATASET.

4.1 Approach 1: FeaturesDataset

In this approach, we work with the concatenating feature vectors from each of the 3 datasets.

- From the first emoticons dataset, we converted the strings of emojis to 13 dimensional lists, then one-hot-encoded them (taking position into account) to obtain 2159 dimensional features.
- From the second deep features dataset, we first linearized the matrices, then applied PCA on them (with n_components = 17) to obtain 17 dimensional embeddings.
- For the third text sequence dataset, we treated each index of the string (0-49) of the 50 length string as a categorical feature (with value ranging 0-9) and performed one hot encoding, getting 50×10 matrices for each input entry. Then we linearized them to obtain 500 dimensional embeddings.

The combined feature vector is then subjected to various dimensionality reduction techniques such as LDA and PCA which we have previously used in Task 1. In this case too PCA works best with n_components = 17.

Various models from the **ML Models** were tried on the PCA applied test set. In case of Neural Networks, the first layer consisted of 32 neurons. The subsequent layers consist of 8,4 neurons while the last layer consisted a single neuron. We used the Relu function for the first two layers and the sigmoid function for the last layer.

4.1.1 Plotting the Results:

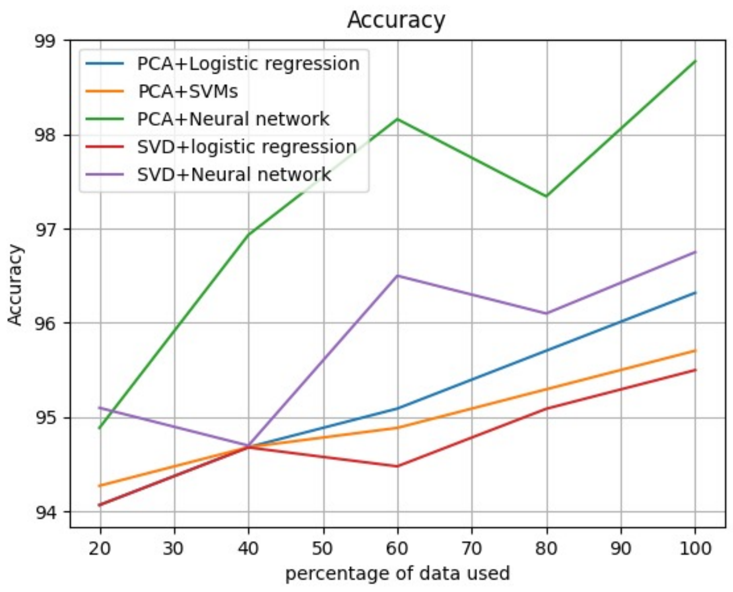


Figure 7: Approach 1

We can see that PCA does a much better job. This is because PCA handles multicollinearity better than LDA. The feature vectors that were combined have high correlation as they are just different representations of the same data. This is why PCA performs better.

4.2 Approach 2: LogitsDATASET

In this approach, we generated the dataset by simply combining the final logits (i.e. probabilities) produced by the respective best models in each dataset from Task 1. Best models for each of the datasets were,

- Emoticon Dataset: SVMs
- Deep Features Dataset: PCA followed by Neural Network
- Text Sequence Dataset: CNN

4.2.1 Data Pre-Processing

In this dataset, we have 3 dimensional features for each input entry, with values between 0 and 1 (since they are, after all, probabilities). First we standardized (**Z-score normalization**) the dataset using *StandardScaler()* from *sklearn.preprocessing library*.

4.2.2 Hyper-parameters

We have then tried fitting various ML Models as mentioned in the ML Models section. We have used the following hyper-parameters,

- **Deep Learning:** A 5-layer densely connected neural network with 9,849 trainable parameters, consisting of 256, 32, 16, 4, and 1 neuron per layer. The first four layers use the ReLU activation function, while the final layer uses sigmoid. Dropout layers with a 0.2 rate are applied between each dense layer for regularization.
- **XGBoost:** max_depth: 10, learning_rate: 0.9714157383088736, subsample: 0.5563695074973118, 'colsample_bytree: 0.8143389859033499

4.2.3 Plotting the Results

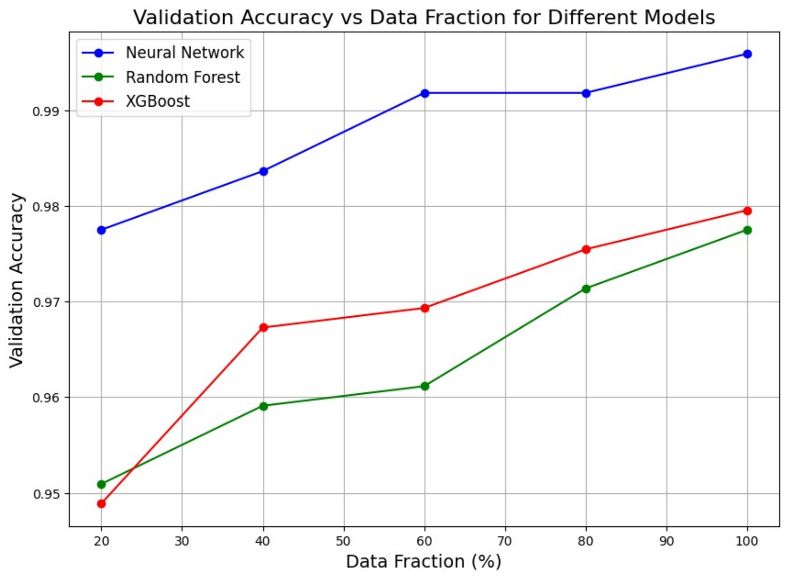


Figure 8: Approach 2

4.3 Results

The best accuracy was obtained for **Logits Dataset (Approach 2)**, with **Neural Networks**. Final accuracies we got were - We This might be because it can learn complex associations from the given data and it is especially suited for features with numerical values (array-type numerical data), as is the case here.

5 Conclusion

In conclusion, this project explored a comprehensive analysis of three distinct datasets using multiple machine learning models, with the goal of determining the best-performing binary classifier for each dataset. By evaluating model performance based on both validation accuracy and training data size, we identified effective approaches for binary classification across different data representations: emoticons (SVM), deep features (PCA + NN), and text sequences (CNN).

Task 1 demonstrated the importance of selecting models that balance accuracy and computational efficiency, while Task 2 showed how combining feature representations from multiple datasets can potentially enhance model performance. Our findings indicate that, combining datasets offers a similar accuracy to accuracy on just the Deep Features dataset.

Overall, this project highlighted the challenges of high-dimensional data, feature engineering, and model selection, offering valuable insights into the development of efficient machine learning models for binary classification tasks. Through the collaborative efforts of our team, we successfully completed the project objectives, and the results provide a strong foundation for further exploration of similar tasks in machine learning.

Acknowledgments

We would like to express our gratitude to Prof. Piyush Rai for the guidance and valuable insights provided throughout this project. We would also like to thank the teaching assistants for their assistance with the datasets and experiments. Additionally, we are grateful for the opportunity to work with the datasets provided for this project. Lastly, we appreciate the collaborative effort of all team members involved in analyzing the datasets, developing models, and writing the report.

References

[1] Geeks for Geeks. Managing high dimensional data in machine learning. `https://www.geeksforgeeks.org/managing-high-dimensional-data-in-machine-learning/`, 2023.

[2] Jonathan Hui. Machine learning: Singular value decomposition (svd) & principal component analysis (pca). `https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pc` 2020.

[3] Scikit-learn Developers. Scikit-learn: User guide. `https://scikit-learn.org/stable/user_guide.html`, 2024.

[4] TensorFlow Developers. Tensorflow keras api documentation. `https://www.tensorflow.org/api_docs/python/tf/keras`, 2024.

[5] XGBoost Developers. Xgboost documentation. `https://xgboost.readthedocs.io/en/latest/index.html`, 2024.

[3] [4] [5] [2] [1].