Binary Classification Project

Krish Arora
230571.

Rohan Garg
230863

1 Task 1

Task 1 involves creating a machine learning model for binary classification on 3 different types of dataset

1.1 1st Dataset

Following are the approaches we explored to solve the given problem. The final method proved to be the most effective, delivering the best results.

1. Simple extraction of features by converting each emoji to its unicode and then applying simple models like SVM , Logistic Regression, Decision trees. But this could not give validation accuracy more than 53%.
2. Then we analyzed the distribution by plotting it using matplotlib module, taking two features at a time(as it is not possible to visualise more than 2 features on a 2D screen) .The attached **Folder** contains visualisation of 2 features at a time; the plot figure p and q represent feature p plotted on x axis and feature q plotted on y-axis(plotting code - plotting.py). We analyzed and concluded from the plot that the distribution is actually not seprable like this because it was completely symmetric i.e. if there is a data point in class 0 then there is also a data point in class 1 nearby. So this method of feature extraction didn't work.
3. Next we tried feature extraction using pre trained embeddings for emoji data found on github as word2vec does not contain embeddings on emoji data. and by that method I did feature extraction and trained neural networks on that but that gave me **80 percent** accuracy which is still less because that model was trained on data from social media but I got the glimpse that if I used a better feature embedding than I would get a good accuracy.
4. Then I learned 7 dimensional embeddings(optimized using validation dataset) using the training data and now as I have trained embeddings specifically for that kind of data then training a neural network on it gave a good validation accuracy **above 95 percent** but this became a very complex model.
5. But after flattening the previously learned embeddings (now each input has 13*7 features) and then using a Linear SVM gave an accuracy around **97-98 percent** i.e. after flattening the dataset was linearly seperable.

END MODEL USED- Feature extraction with data and then SVM

- 20% training examples: 96%
- 40% training examples: 97.1%
- 60% training examples: 97.5%
- 80% training examples: 98.3%
- 100% training examples: 98.1%

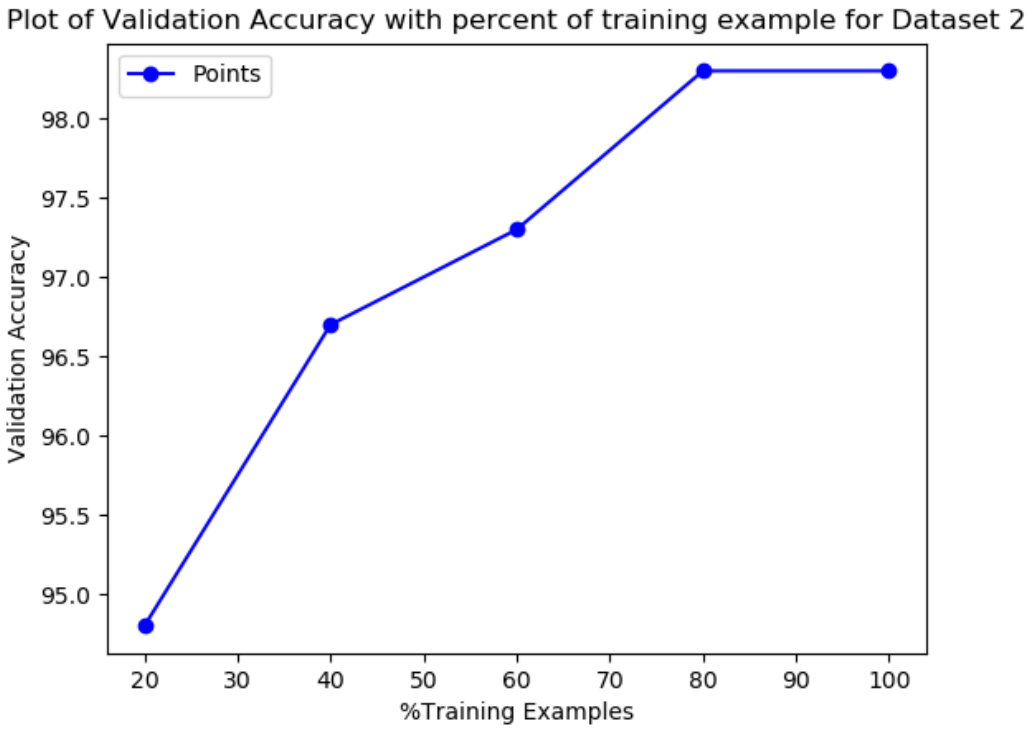

Figure 1: Enter Caption

1.2 2nd Dataset

1. First I tried flattening each input and then fitting a SVM on the 13*768 = 9984 (less than 10000) dimensional data using RBF kernel. Training and testing took around 1.5 minutes and gave **96 percent validation accuracy**

2. Then training a linear SVM on the dataset gave around **98 percent accuracy** and took around 30 seconds for training.

3. Then I thought that if the data is linear why not to use a simple linear regression model i.e. Logistic Regression which gave a 98 percent validation accuracy in just 10 seconds of time, so this is the best model we have.

END MODEL USED - Flattening then Logistic Regression

- 20% training examples: 94.8%
- 40% training examples: 96.7%
- 60% training examples: 97.3%
- 80% training examples: 98.3%
- 100% training examples: 98.3%

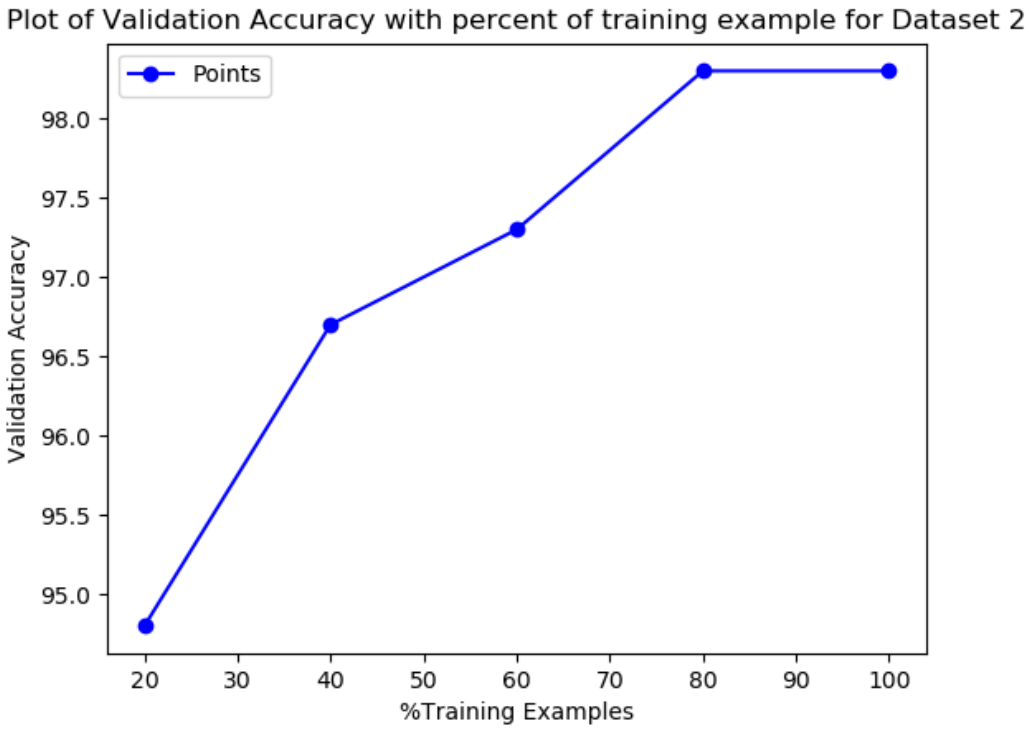

Figure 2: Enter Caption

1.3 3rd Dataset

The provided dataset contains 7,080 training examples, each representing as a string of integers (0-9) with a length of 50. Treating the numeric string as a feature vector with each element of a string as one feature and training binary classifiers like SVM's, Xgboost and even neural networks could not give a validation accuracy of more than 60Then I treated the dataset as a sequential data of 50 timesteps and created a neural network to understand the sequence. This model gave a validation accuracy around 72%.

But then using a 1D convolution layer to capture local dependencies in the sequence gave a boost to accuracy which reached around 84-85%. This is the best model we could make. Thus dataset has been transformed into a sequential format with 50 timesteps, resulting in a matrix of shape (7080, 50). Both validation and test datasets are transformed in a similar way.

A deep learning architecture is developed using the Keras Sequential model, which includes the following components:

- Embedding Layer: Converts each integer token into a 16-dimensional vector representation, capturing semantic information rather than using sparse one-hot encoding. The input dimension is 10, corresponding to the integer tokens from 0 to 9. Without embeddings the model gave a validation accuracy of
- 1D Convolution Layer: Applies 64 filters with a kernel size of 3. This layer captures local dependencies in the sequence, analogous to how 2D convolution extracts features from images and captures spatial information. The ReLU activation function introduces non-linearity, enabling the model to learn complex patterns.
- LSTM Layer: This layer retains an internal state to capture long-range dependencies and sequence information, which is essential for understanding patterns across timesteps in sequential data.

- Dense Layer: A dense layer with 16 units (using ReLU activation) further transforms the features.

- Output Layer: The final dense layer consists of a single unit with sigmoid activation for binary classification.

The model achieved a validation accuracy of 80% after approximately 10 epochs. After training for an additional 7-8 epochs, the validation accuracy reaches around 86%, with training accuracy of about 88%, after which model starts showing overfitting.

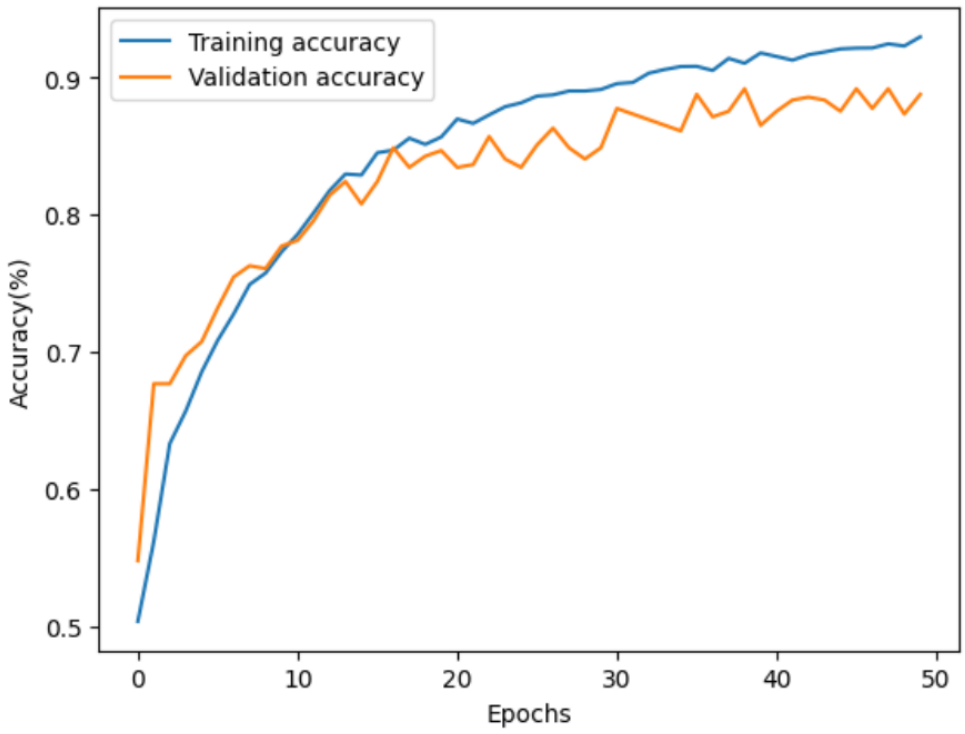

Figure 3: **Accuracy vs Epochs**

Further analysis shows that training with a fraction of the training examples yields the following validation accuracies:

- 20% training examples: 77.71%

- 40% training examples: 79.14%

- 60% training examples: 80.37%

- 80% training examples: 83.44%

- 100% training examples: 86.71%

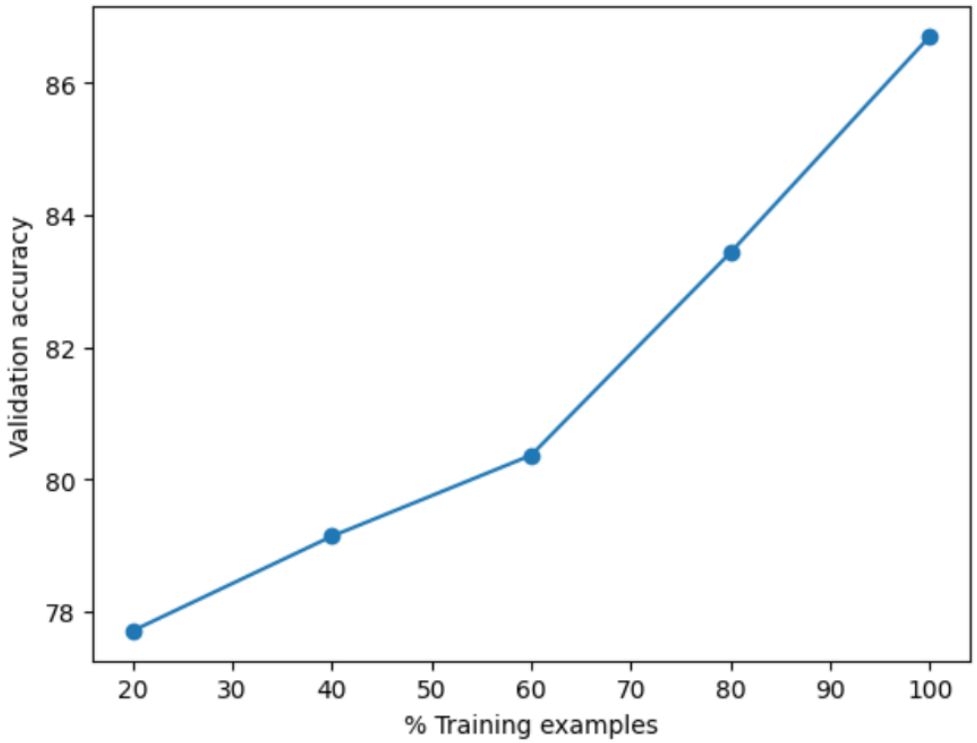

Figure 4: **Validation Accuracy vs. Fraction of Training Examples**

2 Task II

In Task 2 we have combined the predictions of three models. We took predictions from each dataset and found the mode of predicted labels from each model(hard voting) i.e. if two models output 1 and the other one outputs 0 it would give 1 as prediction. On analysis it shows that accuracy increases a little bit for 100% training data. Major difference is seen accuracies when a fraction of training examples is used i.e. even with lower percentage of training data higher validation accuracy is achieved. Furthur analysing by training each model only on a fraction of training examples and then combining the results shows following results:

- 20% training examples: 95.5%
- 40% training examples: 97.34%
- 60% training examples: 97.75%
- 80% training examples: 97.98%
- 100% training examples: 98.66%

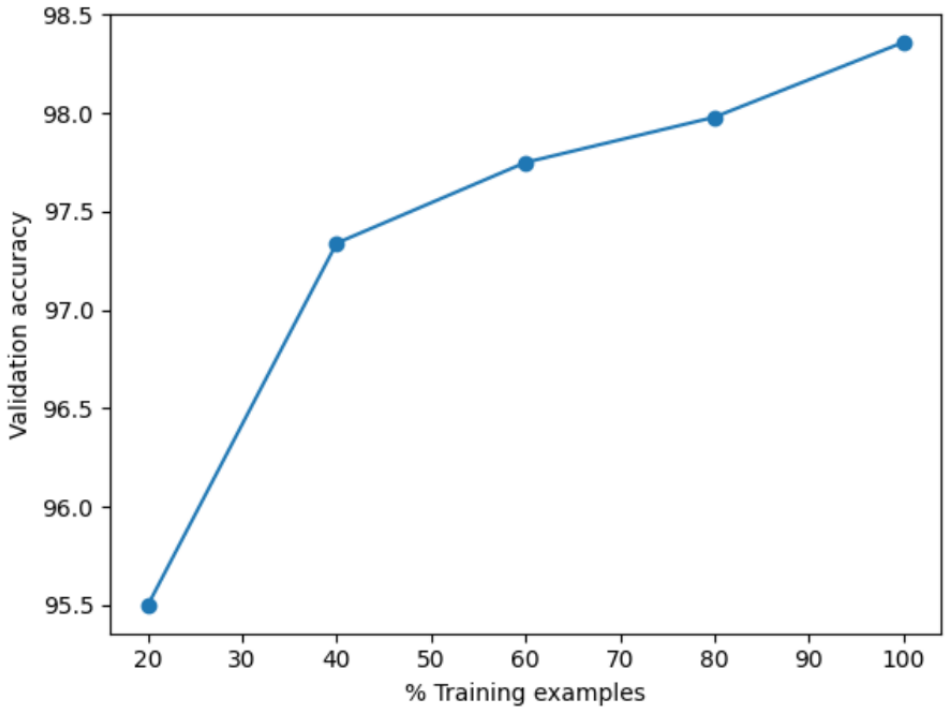

Figure 5: Validation accuracy vs fraction of training examples

Acknowledgments

...

References