CS771 - Intro to ML (Autumn 2024): Mini-project 1

ML Group 24
Nishvaan Sai H - 200648
Krishna Kumar Bais - 241110038
Sevak Baliram - 241110065
Rohan - 241110057
Amit Kumar Meena - 220126

October 22, 2024

1 Introduction

We are given three binary classification datasets, each representing the same machine learning task and generated from the same raw dataset. The three datasets only differ regarding features representing each input from the original raw dataset. These three datasets further consist of training, validation, and test sets.

1.1 Objective

1. **Task 1:** The first task is to train some binary classification models on each of the three datasets and pick the best possible binary classification model for each.

2. **Task 2:** The second task is to train some binary classification models, using all three datasets combined, and pick the best possible binary classification model.

2 Task 1

In this task, we analyzed the given datasets to find a deeper meaning of the features and used the datasets individually to train models and find the best possible solution.

2.1 Emoticon as Features Dataset

This dataset has 13 emoticons as a feature for a given sample. Example of a sample feature is as follows(in the next page):-

😮 😛 😣 🛐 🚭 🚼 🖋 😑 😣 🖋 😹 😣 /

2.1.1 Data Analysis

With the emoticon dataset first, we separated the emoticons into 13 features and then did Exploratory Univariate Data Analysis; we found out that for each of the first eight features, the emoticons

😛 🛐 😑 😣 🚼 🖋

are frequently occurring, i.e., for each of the first 8 features, the emoticons mentioned above are each present in about 1/7th of the sample. The remaining 1/7th of the sample contains unknown emoticons (which don't occur as frequently) for each feature. Similarly, for the last five features, the emoticons

😑 😣 🖋 /

are frequently occurring, i.e., for each of the previous five features, the emoticons mentioned above are each present in about 1/5th of the sample. The remaining 1/5th of the sample contains unknown emoticons (which don't occur as frequently) for each feature. Then we looked at the dataset in Excel and found a pattern that in a given sample for the first eight emoticons, the emoticons

😛 🛐 😑 😣 🚼 🖋

are always present accompanied by two other varying emoticons. So, for a given sample, the first eight emoticons are present as a permutation of 6 known emoticons

😛 🛐 😑 😣 🚼 🖋

and two varying emoticons. Similarly, in a given sample for the last five emoticons, the emoticons

😑 😣 🖋 /

are always present, accompanied by one other varying emoticon. So, for a given sample, the last five emoticons are present as a permutation of 4 known emoticons

😑 😣 🖋 /

and one varying emoticon.
This led to the conclusion that the dataset can be considered a text-data featured dataset, with each sample containing 13 words, where each emoticon is handled as a word.

2.1.2 Feature Transformation

The dataset is given as a string with 13 characters or emoticons. We then transformed the feature into a string of 13 words separated by space, where each word is the emoticon's ordinal value (using the in-built 'ord' function in Python). This was done to make it more presentable as a text-data feature.
For example: '128558 128539 128547 128720 128685 128668 128559 128529 128547 128559 128569 128529 128572' is the updated string for the below emoticons-
.

😮 😛 😣 🛐 🚭 🚼 💉 😑 😣 💉 😹 😣 /

2.1.3 Feature Encoding

For this, we tried various encoding methods like Bag of Words (BoWs), One-Hot encoding, and Text embeddings, and using cross-validation with the validation set, we narrowed it down to Text-embedding.

The Text embedding was done in the following way:

1. **Tokenisation of Text:** The text data is divided into individual tokens (words). Each sequence of text is split into separate tokens using '.split()'. Example: '128558 128539 128547 128720 128685 128668 128559 128529 128547 128559 128569 128529 128572' is converted to ['128558', '128539', '128547', '128720', '128685', '128668', '128559', '128529', '128547', '128559', '128569', '128529', '128572'].
2. **Vocabulary Creation:** A vocabulary is created by extracting all unique tokens (words) from the training and validation datasets. These tokens are then mapped to unique integer indices, starting from 1.
3. **Tokenisation of Sequence:** After creating the vocabulary, each tokenised sequence is converted into its numeric equivalent by replacing each token with its corresponding index from the vocabulary.
4. **Embedding Layer:** Now we have defined an embedding layer which converts the integer representation of words into dense vectors of fixed size, known as embedding vectors. This layer learns the representation of each word in a continuous vector space during training. This is done using the library **tensorflow.keras**. Here, the input dimension is the vocabulary size, i.e., the number of unique tokens in your data. This is the range of the integer values that represent your tokens. The output dimension (which specifies the dimensionality of the vector space into which the tokens will be embedded) used is 16 (which was decided after cross-validation).
5. **Model Architecture:**
   (a) **LSTM Layer:** This layer processes the sequence data using a Long Short-Term Memory (LSTM) network, which helps capture temporal dependencies between tokens in the sequence.
   (b) **Dropout Layer:** Dropout is applied to prevent overfitting, randomly setting some outputs from the LSTM to zero during training.
   (c) **Flatten Layer:** The output of the LSTM is flattened into a single long vector, which can be passed to a dense layer.
   (d) **Dense Layer:** This fully connected layer has eight units and a ReLU activation function. It reduces the dimensionality and applies a non-linear transformation.
   (e) **Output Layer:** Finally, the output layer is a single neuron with a sigmoid activation function suitable for binary classification (0 or 1).
6. **Training the Model** This is also done using **tensorflow.keras**. The model is trained using the binary cross-entropy loss and Adam optimiser, with accuracy as a metric.
7. **Note:** Every hyperparameter here is chosen so that the number of trainable parameters is well below 10,000 and provides the best accuracy while cross-validation using the validation set.

2.1.4 Feature Transformation

After embedding the feature, we are left with a matrix of shape (13,16) for each sample. We will now flatten the matrix and have 208 features for each sample.

2.1.5 Feature Scaling

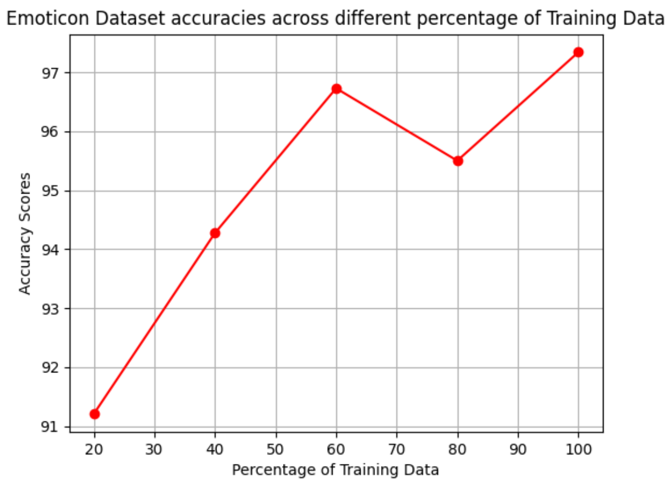
Many feature scaling methods were used, and standardization provided the best results in cross-validation checks with a validation set. Here, the standardization is done using **Scikit-learn**.

2.1.6 Model Selection and Training

After all the above feature processing, various models such as Logistic Regression, Support Vector Machine (SVM), k-Nearest Neighbors (k-NN), Random Forest Classifier, Decision Tree Classifier, AdaBoost Classifier, and XG-Boost Classifier were used for the binary classification problem. Here, all the models were applied using the library **Scikit-learn**.
SVM provided the best results from all the models used while checking with the validation set. Further hyperparameter tuning was done to avoid overfitting and give the best results.Thus, we train the SVM model using the training set.
Disclaimer: For hyperparameter tuning across the entire model training on the emoticon dataset and the model selection, the best model was selected based on which values and models provided the best accuracy across the validation set when only considering 20% of the training set.

2.1.7 Variation of accuracies in various fractions of the Training set

After training the model using the training data set, we predicted the labels for the validation set. We then compared it with the labels provided for the validation set and found the accuracy, which was also done using the **Scikit-learn** library in Python. We also checked the confusion matrix for better prediction details (also used **Scikit-learn**).
We then checked how the model accuracy varies along various percentages(20%, 40%, 60%, 80%, 100%) of training sets by splitting the training set(this is done using **Scikit-learn**) right after importing the data and doing all the processes mentioned above. The image below shows the accuracy variations.



2.1.8 Test Dataset Prediction

Apply the same feature transformations, encoding and scaling, and predict using the model you trained for 100% of the Training set.

2.2 Deep Features Dataset

Here, the feature of each sample is represented by a matrix of shape (13,768).

2.2.1 Feature Transformation

We flattened the matrix dataset so that each sample now has 9984 features. After that, we used the log Function Transformer from **Scikit-learn** to normalise the data.

2.2.2 Feature Reduction

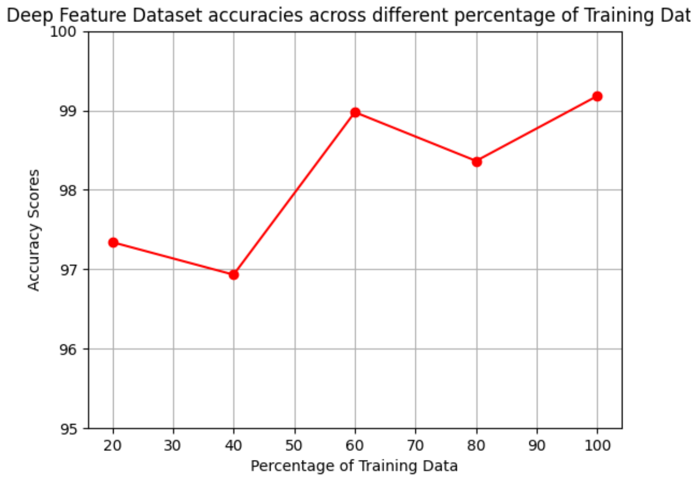
Since the no. of features is huge (almost close to 10,000),we have reduced the number of features using Principle Component Analysis(PCA) using the Python library **Scikit-learn**.We have reduced it to 100 features, and this hyperparameter value was decided through cross-validation with the validation set.

2.2.3 Model Selection and Training

After all the above feature processing, various models such as Logistic regression, Support Vector Machine(SVM), k-nearest-neighbour (k-NN), Random Forest Classifier, Decision tree Classifier, Adaboost classifier, and XG-Boost Classifier, were used for the binary classification problem. Here, all the models were applied using the library **Scikit-learn**
SVM provided the best results from all the models used while checking with the validation set. Further hyperparameter tuning was done to avoid overfitting and give the best results.
Thus, we train the SVM model using the training set.
Disclaimer: For hyperparameter tuning across the entire model training on the emoticon dataset and the model selection, the best model was selected based on which values and models provided the best accuracy across the validation set when only considering 20% of the training set.

2.2.4 Variation of accuracies across various percentages of the Training set

After training the model using the training data set, we predicted the labels for the validation set. We then compared it with the labels provided for the validation set and found the accuracy, which was also done using the **Scikit-learn** library in Python. We also checked the confusion matrix for better prediction details (also used **Scikit-learn**).
We then checked how the model accuracy varies along various percentages(20%, 40%, 60%, 80%, 100%) of training sets by splitting the training set(this is done using **Scikit-learn**) right after importing the data and doing all the processes mentioned above.
The image below shows the accuracy variations.



2.2.5 Test dataset predictions

Apply the same feature transformations, encoding and scaling, and predict using the model you trained for 100% of the Training set.

2.3 Text Sequence Dataset

The features of each input are given in the form of a length 50 string consisting of 50 digits.

2.3.1 Data Analysis

From the emoticon dataset, we know that in a sample, the first eight emoticons always have six known emoticons and two varying emoticons, and there are always four known emoticons and one varying emoticon in the last five.
So, while we were analysing the text sequence dataset, we found for every sample, substrings ‘15436’, ‘464, ’422’, ‘1596’, ‘614’, ‘262’, and ‘284’ are always present in the feature string.
Additionally, all these substrings are non-overlapping, and three occur twice: ‘1596’, ‘614’ and ‘262’. So, we have ten non-overlapping substrings always present in a sample feature. The same was true for all the samples when comparing the relative positions of the feature’s substrings with the emoticons’ relative position in the emoticon dataset. From this, we have concluded that the substrings ‘15436’, ‘464, ’422’, ‘1596’, ‘614’, ‘262’, and ‘284’ are different representations of the emoticons

😛 🛐 🚼 😑 😣 🖋 /

respectively. The varying emoticons can also be represented as a string, which is found as a substring in the text sequence sample feature.

Another observation is that all the '0' at the start of the string are just padding added so that the sample feature will be a size 50 string. Also, the varying emoticons string representation should always be of length greater than or equal to three, and the string shouldn't start with '0'.
From all these observations and conditions, we separated the substrings to form a 13-featured dataset, where this dataset has a bijective mapping to the emoticon dataset, where instead of emoticons, we have strings.
Example: '00001543646427181596614280026242231322841596262614' sample is converted to ['15436', '464', '2718', '1596', '614', '2800', '262', '422', '3132', '284', '1596', '262', '614'].

2.3.2 Feature Transformation

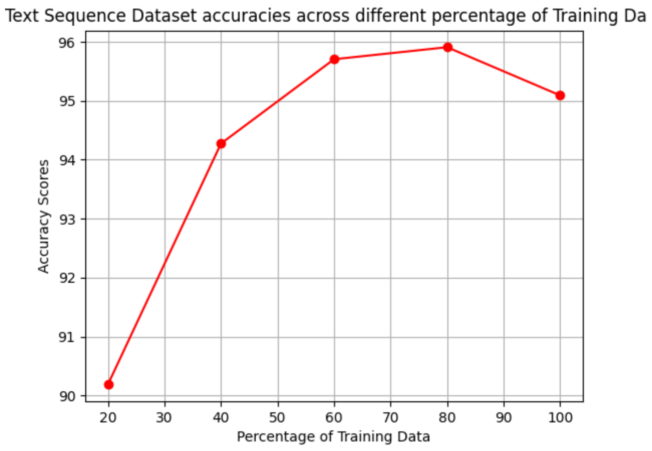
Convert the 13 features into a single feature, as a string of 13 words
Example: ['15436', '464', '2718', '1596', '614', '2800', '262', '422', '3132', '284', '1596', '262', '614'] sample is converted to "15436 464 2718 1596 614 2800 262 422 3132 284 1596 262 614"

2.3.3 Feature Engineering, Model Selection and Training

Since the feature representation is the same as the one in the emoticon dataset, the whole model architecture followed by the emoticon dataset will be applied to the text sequence dataset after transformation, as they are virtually the same.

2.3.4 Variation of accuracies in various fractions of the Training set

After training the model using the training data set, we then found the accuracy on validation set, which is also done using the **Scikit-learn** library in Python. We also checked the confusion matrix for better prediction details (also used **Scikit-learn**). We then checked how the model accuracy varies along various percentages(20%, 40%, 60%, 80%, 100%) of training sets by splitting the training set(using **Scikit-learn**) after importing the data and doing all the work mentioned above. The image in the next page shows the accuracy variations.



3 Task 2

3.1 Combined Feature Dataset

Here, we will combine all three of the datasets mentioned above. Since the emoticon and the text sequence dataset are virtually the same, we will only combine the emoticon and deep feature datasets.

3.1.1 Feature Engineering

The emoticon and deep feature dataset feature engineering will be done the same way as mentioned previously.

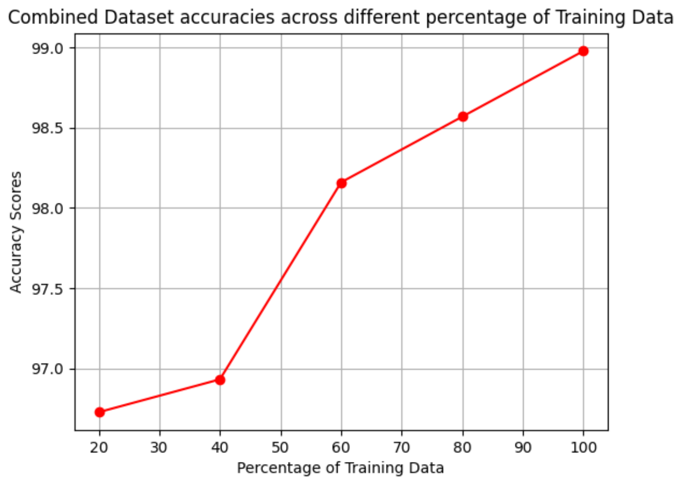
3.1.2 Combining Features

Various methods were analysed to combine features. They are Concatenation of Features, Stacking or Ensemble, and Feature averaging. The Stacking or Ensemble method was chosen based on cross-validation with a validation set.
Here is how stacking or ensemble is done:

1. **Train Base Models:** We used the models already trained for the two datasets. We trained the emoticon dataset with SVM, and the hyperparameters were fine-tuned, as mentioned in part 2.1.6. Similarly, we trained the Deep Features dataset with SVM, and the hyperparameters were fine-tuned, as mentioned in part 2.2.3. Here, the models were used from the **Scikit-learn** library in Python, and the model is trained using the processed training set.
2. **Create meta features:** Instead of predicting the labels, predict the probability of the dataset for each of the trained models and stack them together as two columns. In this way, we will get our new meta-features.
3. **Meta-model selection and training:** Various models were tried to train the meta-features: SVM, Logistic Regression, and Random Forest Classifier.Of these models, the Random Forest classifier provided the best accuracy with the validation set. Here, all the models were accessed through **Scikit-learn**.

3.1.3 Variation of accuracies across various percentages of the Training set

After training the model using the training data set, we predicted the labels for the validation set. This is done by first converting the validation set features into meta-features and using the meta-model to predict the labels. We then compared the predicted labels with the labels provided for the validation set and found the accuracy, which was also done using the **Scikit-learn** library in Python. We also checked the confusion matrix for better prediction details (also used **Scikit-learn**).
We then checked how the model accuracy varies along various percentages(20%, 40%, 60%, 80%, 100%) of training sets by splitting the training set(this is done using **Scikit-learn**) right after importing the data and doing all the processes mentioned above.
The image below shows the accuracy variations.



3.1.4 Test Dataset Prediction

Apply the same feature transformations, encoding and scaling, and predict using the base and meta-models you trained for 100% of the Training set.

4 Comments on the accuracy of all four models

1. The deep-featured dataset model provided the best accuracy among all four models. It provided an accuracy of 99.1820
2. The combined feature dataset model provided the second-best accuracy of 98.9775%; it improved from the emoticon and text sequence dataset, but we couldn't improve the accuracy further.
3. Both the emoticon and text sequence-based datasets provided 95-96% accuracy.