Assignment 8:

ML project for abusive text detection.

March 22 - March 31

Hate speech classification using TF-IDF+KNN, LSTM, and Transformer

In this assignment, you will learn how to classify comments as abusive or non-abusive using the TF-IDF feature extraction technique and KNN classifier. In part-2 of the assignment, you will classify the same thing using LSTM. And in part-3, you will build a text classifier using Multilingual language models like mBERT and MuRIL.

Dataset:
The dataset contains a collection of comments in the Hindi language. Based on their content, the comments have been labelled as either abusive (0) or non-abusive (1). The dataset has around 20,000 comments, with comments labelled as abusive and non-abusive. You are given the dataset for training and validation in the google drive link, download the datasets and train your models(you need to split it to make a separate train and validation set). Test-set is hidden to evaluate the performance of your model.

Data Fields:

- Comment Text: The text content of the comment.
- Label: A binary label indicating whether the comment is abusive (0) or non-abusive (1).

Task 1[50 Marks]: In this task, you have to extract features from the dataset using *TF-IDF (term frequency-inverse document frequency)*, then do classification using the KNN classification model.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc. (Note that the dataset will contain emojis that may be

useful. In that case, it is advisable not to remove the emojis; instead, use them to classify correctly.)

2. Use TF-IDF for feature extraction and create a feature matrix.
3. Using a validation set, guess the optimal 'K' for your KNN classifier. You can also play with other parameters of the algorithm to choose the best-performing one for your dataset.
4. For each example in the train data
   a. Calculate the distance between the query example(test instance) and the current example from the train data.
   b. Add the distance and the index of the example to an ordered collection.
5. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances.
6. Pick the first K entries from the sorted collection.
7. Get the labels of the selected K entries.
8. and return the mode of the K labels.

Task 2[50 Marks]:

As you may have guessed, the above approach is fast but may not be optimal for text input. One of the main reasons is that in the above technique, you are not considering the word ordering in a sentence which can be crucial for text classification.
So you will use LSTM(a variant of RNN) to catch the ordering and dependencies among the words in a sentence and classify them.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc. (Note that the dataset will contain emojis that may be useful. In that case, it is advisable not to remove the emojis; instead, use them to classify correctly.)
2. Create the vocabulary of your model. (Fun exercise: You can change the size and check the performance vs efficiency trade-off.)
3. Define the maximum sequence length of your dataset.
4. Accordingly, tokenise and pad/truncate the sentences in your dataset.

5. Split it to train and validation set. (Preferably, the ratio of split should be 80-20).
6. Define Dataloaders for batch processing while training.
7. Define your model class. It should include the definition of the embedding layer, LSTM layer, along with classification layer, Dropouts, and activation functions.
8. Train the model on the training data given.
9. Find validation set performance at the end of each epoch.
10. If validation set performance doesn't improve over k iterations (called patience), stop training(early stopping).
11. Load best model weights based on the validation set performance.
12. Find performance on the test set(test dataset will not be shared, we will evaluate your model using the test set).

Task 3[50 Marks]:

In this part, you will use state-of-the-art transformer-based models mBERT and MURiL for text classification. Here mBERT is pre-trained using 104+ language data, and MURiL is pre-trained with 17 in Indian languages.
The procedures are more or less the same as above.

Sample steps of the expected algorithm:

1. Preprocess the data, remove unnecessary symbols, stop words, etc. (Note that the dataset will contain emojis that may be useful. In that case, it is advisable not to remove the emojis; instead, use them to classify correctly.)
2. Load the Model & Tokenizer from the huggingface library.
3. Design the dataloader class for tokenising and batch process the input data.
4. Create a model class; here you can directly use the classification model from the huggingface library or use the normal model without a classification head and add your own

classification layer above it. Also, the dropouts, and activation functions should be defined here only.

5. Design the train loop, and define the optimizers, schedulers, and loss functions.
6. Split the data to train and validation set. (Preferably, the ratio of split should be 80-20).
7. Train the model on the training data given.
13. Find validation set performance at the end of each epoch.
14. If validation set performance doesn't improve over k iterations (called patience), stop training(early stopping).
15. Load best model weights based on the validation set performance.
16. Find performance on the test set(test dataset will not be shared, we will evaluate your model using the test set).

Submission guidelines:

1. For implementing TF-IDF, you can use Scikit Learn's TfidfVectorizer. Also, to implement the KNN classifier, use the Scikit Learn's KNeighborsClassifier.
2. For LSTM, use the torch.nn package.
3. For task 3, you are allowed to use the 'transformers' package.
4. Run the code using google colab.
5. We will release the test data 30 min before the submission deadline, and you need to run a cell at the end of your notebook and report the test accuracy and macro-f1 score there only.
6. Also, add a function which will calculate the intersection(number of common sentences) between the test set with the train and the validation set. Run the function at the end, and print the results. **Without this cell, we will not grade your submission.**
7. You don't need to submit the whole code; just share the google colab link with us through moodle submission link.

8. No partial marks will be given, marks will be relative as per the accuracy, and macro-f1 score reported by the whole class. So try to play with the parameters, extra layers, data processing to get the most out of it.
9. Also, submit a report in pdf format where you will write a short summary about your learning while doing this assignment, not more than 500 words.