Sentiment Analysis of Code-Mixed Data

by Lakshay Piplani (PSU ID: 9 5343 5859 ; LJP5718@PSU.EDU)
and Durva Dev (PSU ID: dbd5616)
Github Repository Link

1 Problem Definition.

Sentiment Analysis on Code-Mixed Data

In this project, we perform sentiment analysis on a dataset containing words in three different languages: English, and two Indian languages, namely, Bengali, and Hindi. Such a dataset containing sentences formed from different languages is called code-mixed data. Below we breakdown the important components of this project.

1.1 What is Code-Mixed Data?

The use of linguistic elements - words, phrases and clauses from more than one language in a single sentence or text is called **code-mixing** [6]. For example, below sentences contain words from English, Bengali, and Hindi [7].

Example 1: this is a joke ডাউনলোড করার চেষ্টা করছি और काम करो be nice get টাকা ফেরত কূড়া
Example 2: शायद यह मेरा है দক্ষতার অভাব কিন্তু I find it quite complicated

Note that there are other versions of code-mixed data where all words are written in the Latin script (commonly, the English alphabet). However, we have used data where each word is written in its native script.

1.2 What are the challenges with Code-Mixed Data?

Below are some common challenges faced during processing of code-mixed data:

- Lack of proper grammar: Code-mixed data is often found in informal setting such as social media, where grammatical rules are frequently violated. Moreover, mixing words from different languages makes it hard to consistently adhere to grammar rules of any one language.
- Lack of high-quality clean data: Due to the complex nature of code-mixed data (spelling and grammar variations), it is hard for human annotators to correctly label code-mixed sentences.
- Complex Embedding structure: We cannot naively use embeddings from separate language models trained on different languages. We have

to either train our own custom embeddings for the specific task or use embeddings from a model trained collectively on all the languages found within the corpus to accurately apture relationships between words/tokens.

1.3 Existing Approaches to Sentiment Analysis of Code-Mixed Data

Prior works on this topic have different in the embeddings as well as the model architectures they have used.

- Embeddings used: TF-IDF and GloVe [6], mBERT [2]. Some researchers have proposed custom multilingual models for specific languages such as [7]. These models can be used for retrieving embeddings for code-mixed data

- Non-transformer neural network architectures used: Classical algorithms like Support Vector Machines (SVM), ensembles of Random Forests, Logistic Regression, and SVMs, LSTM-based Recurrent Neural Networks (RNN) [6], Convolution Neural Networks (CNN) and combinations of Convolution and LSTM layers [4]

1.4 Motivating Research Literature

The research paper [4] that motivated us to take up this task for our mid-term project proposed a character-level embedding that was convoluted to form sub-word level representation.
Moreover, they proposed a neural network architecture that combined convolution layer followed by LSTM layers. Therefore, we have tried to emulate their neural network classification architecture, as we describe Section 3.
There are two main points of difference in our work from this base literature. First, the dataset used in this paper is (a) limited in size (<5000) and (b) contains non-English words written in the English alphabet instead of native script. We searched for a larger dataset consisting of words written in native script. Finally, we have developed our model on a much larger dataset that contains non-Latin characters as well. The dataset is described in detail in Section 2 below.
Second, the authors in the motivating paper have trained their own custom character-embeddings. On the other hand, we have explored utility of

pre-trained multi-lingual embeddings by utilizng them as-is/fixed as well as fine-tuning them.We have described the embeddings we used in Section 3 below.

1.5 Scope of Our Work

For this project, we follow an empirical approach to compare performance of different embeddings and neural network architectures for the task of sentiment analysis on code-mixed data. Specifically, we run experiments based on the following dimensions:

- Embeddings
  - Pre-trained embeddings without fine tuning
  - Pre-trained embeddings further fine-tuned
- Neural Network Architectures
  - Combination of Convolution and LSTM layers

These dimensions capture the various non-transformer approaches used by prior works for this task.

2 Dataset Curation

We have used a publicly available tri-lingual (English, Hindi, and Bengali) code-mixed data. It is one of the largest datasets we came across for sentiment analysis. The authors [7] procured this data by obtaining 100,000 samples of Amazon reviews and labelled their sentiment based on the corresponding ratings: greater than 3 stars received Positive label, exactly 3 stars as Neutral, and less than 3 stars as Negative. The dataset is well-balanced, with $\frac{1}{3}$ of all data points belonging to each of the three classes. However, since it is a synthetically available dataset, the quality of the data may not be as high as manually annotated datasets.

Label	Text
Negative	উত্‌পাদ নিরাপত্‌তা code किया नहीं काम
Positive	কাম আমরা হব all উপায় ইচ্‌ছা পুনর্‌নবীকরণ
Neutral	its okay मैं चाहता हूं এটি হবে display the পৃষ্‌ঠাগুলি ভিন্‌নভাবে

Table 1: Dataset Sample

2.1 Dataset Sample

Table 1 above shows a sample sentence for each of the three classes: Positive, Neutral, and Negative.
NB: Eventhough the original dataset contains 100,000 data points, we have taken **two samples** of size **20,000** and **48,000** to train our models within the **compute constraints**.

2.2 Dataset Statistics

For the 48,000-size dataset, below is the distribution of classes:
Neutral, Frequency: 16044
Negative, Frequency: 15999
Positive, Frequency: 15957

And for the 20,000-size dataset,
Neutral, Frequency: 6636
Negative, Frequency: 6583
Positive, Frequency: 6603

Therefore, the datasets are well-balanced.

3 Embeddings and Model Architectures

3.1 Pre-processing and Embeddings

As seen in literature [5], subword representations tend to perform better than character- and word-level representations. As mentioned in Section 1.4, we have used pre-trained embeddings with and without fine-tuning. To convert words into embedding we have used and compared python libraries - BPEmb

& FastText. These are specifically trained on multilingual data aligning wiht our case. The dimension for word embedding for both is 300.

- BPEmb [3] is a collection of pre-trained subword embeddings for 275 languages based on white pair and coding BPE and trained on Wikipedia.
- Fasttext [1], based on Word2Vec, is a library for efficient learning of word representations, and sentence classifications. Fasttext breaks words into subword n-grams and learns embeddings for the subwords. This allows better embedding generation for OOV words than many pretrained models.

In the approach to train models, **using pre-trained embeddings without fine-tuning**, we conducted two separate experiments, utilizing both BPEMB and fast text in both cases the initial tokenization process remain the same. Each text was cleaned according to standard NLP practices (identifying tokens with indic-nlp-library, removing stopwords in different languages using stopwordsiso library) and the language of every token extracted was identified-English, Hindi or Bengali before further processing. We defined monolingual embedding models so we used 3 models for each experiment. The actual embedding process - generation and use differs as

- For *BPEmb*, words were broken down into subword units. We then mapped each subunit to their corresponding vocabulary IDs based on a predefined vocabulary for each language maintained on their official GitHub repository. Each language has a separate vocabulary where subwords are mapped to unique indices. For each model, there was also a vector file that stored the embedding for each subword in its vocab. We used it to generate the initial embedding matrix.
- For *fasttext*, each word was directly embedded without explicitly breaking it into subwords. Fasttext generated word representations dynamically based on subword information.

For both methods, sentence level representations were obtained by aggregating the embedding of all subwords.

In the alternative approach where we **finetuned the initial embedding layer** of the model after initializing it with the pretrained embeddings, we were only able to use BPEmb to carry out experiments as (to the best of our knowledge) fasttext does not provide an explicit mapping of vocabulary indices in the same way the former does.

3.2 Formation of Train, Validation and Test Datasets

For all our experiments, we used a consistent method to obtain the output array Y. We mapped the sentiments 'Positive', 'Negative' and 'Neutral' to 2, 1 and 0 respectively. To implement this we used a simple dictionary.

The case of Fasttext, is easier, the embeddings were directly fed to the model as the input features.

Whenever we used BPEmb, the vector embeddings were fed to the model as parameters. The feature array is the indices of the subwords. However, this part is tricky - because we stacked the embedding vectors to get the matrix, we also had to add an offset to every index equal to a multiple of the vocabulary size (in our case 10k). This was done to prevent any collisions.

3.3 Model Architecture

- The first layer consists of the embedding layer, which we populate using our pre-trained embeddings. The token-level is sub-word level and each sub-word has embedding size of 300.
- Next, we apply a 1 dimensional Convolution layer with 128 filters, and a kernel size of 3
- This is followed by a 1 dimensional Max Pooling layer of kernel size = 3 and stride = 3
- Then, we feed every sentence, one element at a time (for a total of 300 time steps) to the first LSTM layer, which outputs each time step as a 128-length vector
- This is followed by a similar LSTM layer that has 128 output units. These recurrent layers help to capture long-term dependencies/context within a sentence in an ordered manner
- Then, a final fully connected layer has 3 output units to give the probability of each class

Below is PyTorch's description of the model's layers:

- (conv1d): Conv1d(1, 128, kernel_size=(3,), stride=(1,), padding=valid)
- (maxpool1d): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
- (lstm1): LSTM(128, 128, batch_first=True, dropout=0.2)
- (lstm2): LSTM(128, 128, batch_first=True, dropout=0.2)
- (fc): Linear(in_features=128, out_features=3, bias=True)
- (softmax): Softmax(dim=1)

Furthermore,
Optimizer Used: Adam optimizer with learning rate = 0.001 and weight decay = 1e-6
Loss function: Cross Entropy
Number of epochs = 50. We apply early stopping which stops training if validation loss does not decrease for more than 5 epochs.
Batch size =128
Evaluation metrics: Accuracy and multi class F-1 score (averaged)

4 Experiments, Results and Inference

For all the experiments we have conducted the same architecture mentioned in the previous section has been used.

4.1 Hardware

To conduct all these experiments. We have used MacBook Pro M3 chip with 14 core GPU.

4.2 20k Records

We started by sampling 20K records from the total 100k records and conducted the experiments as follows.

BPEmb with embedding fine tuning. The final F1 score reached 0.32

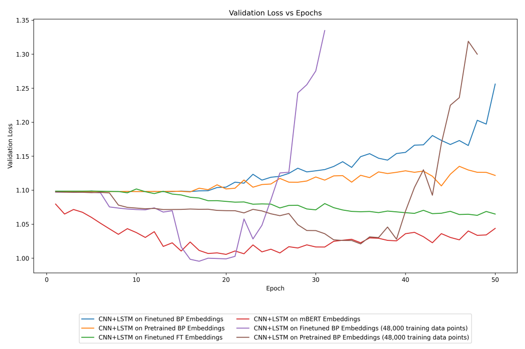

Figure 1: Validation Set Loss vs Epoch

the training log indicates that the model was gradually improving over a box with both training and validation accuracy increasing from the first epoch onwards initially F1 scores for each class was low but they improve tremendously, However it is not a very stable. The training accuracy increased steadily, reaching around 46.5% by epoch 49 and the line validation accuracy fluctuated speaking at 39% in the final EP. The loss decrease was very slow.

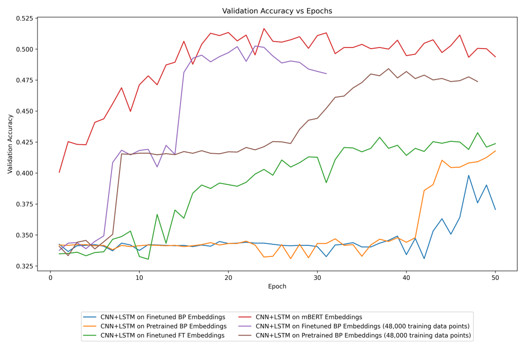

Figure 2: Validation Set Accuracy vs Epochs

BPEmb with embedding without fine tuning. this experiment showed a very similar progression of validation, accuracy and F1 score similar to the model one. The F1 score starts low gradually improving as the pogs go on it reaches its peak validation accuracy of 41% around epoch seven compared to the previous model where it peaks at epoch four with a slightly higher accuracy of 43%.

Clearly, fine-tuning helped the model converge in a steadier manner however, both the models are not stable suggesting over fitting.

Fasttext with embedding without fine tuning. in this model, the F1 scores showed a steady improvement as the previous models the F1 score starts of low and then with a sharp peak around epoch 10 and continue steadily till epoch 50. As shown in the figure this model learns rather quickly, but still isn't as stable. Inferring these results, we decided to experiment with the data set sample and used the second method that is fine-tuning, the pre-trained embeddings as a baseline.

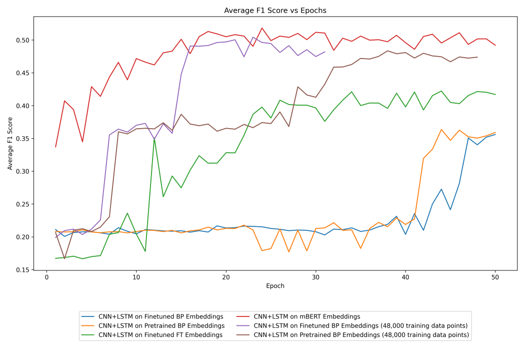

Figure 3: Validation Set F-1 Score vs Epochs

4.3 48k Records

The model in which the embedding are fine tuned reaches a peak test accuracy of 49.26 and a final average F1 score a 49.22. Just like the other models this model also shows in efficiency in the early epoch but still starts converging faster than the 20,000 data points models confirming our thoughts about over fitting. out of all our experiments this is the model that gave us the best accuracy, although it is significantly lower than the transformers. The pre-trained on embedding only model reaches a slightly lower test accuracy of 46.19 and also takes a long time to converge.

4.4 61.5k Records

The models showed a similar performance as that of 48k records but took much longer to train.

4.5 Notes: mBert, Challenges

Since there are no existing base lines for this we also trained mBert (768 embedding size) to **compare** the performance with our models as shown in the figure.

We also did not have the competition resources to test and compare mBert's performance (768 embedding size) on 48K records. we were also not able to accurately train the model on the entire data set that is hundred thousand records. The F1 score peak at 0.2 showing great and inefficiency in training solely because of memory bounds.

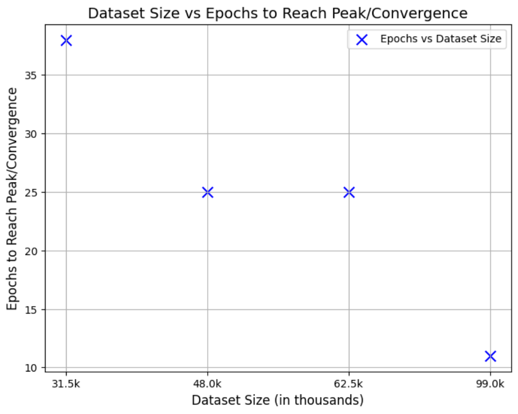

Figure 4: Epochs vs Dataset Size

5 Future Work, Learnings and Experiences

For the future, we can build on this work and improve performance on this task by:

- Collecting higher quality human-annotated data: This can be done by scraping social media websites for multilingual code-mixed data and using manual labor to get high-quality accurate labels

- Changing model architecture and see how it is impacted.

While doing this project, we learned about the ambiguities between context of different languages. It was interesting to see the latest research being done on this topic - how a single model can adapt to lexical and syntactic differences of different languages. While working on tokenization, we got to know about different representations for mixed data and see how multi-lingual tokenization works. We also got a deeper understanding of how embeddings work and how context is transferred in state-of-the-art transformer models. Lastly, we came up with ideas to work within the limits of compute resources available at our disposal. We made several adaptations in terms of limiting data set sizes, simplifying model architectures, and caching embeddings to ensure feasibility of experiments we performed for this project.

References

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, 2019.

[3] Benjamin Heinzerling and Michael Strube. BPEmb: Tokenization-free pre-trained subword embeddings in 275 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[4] Aditya Joshi, Ameya Prabhu, Manish Shrivastava, and Vasudeva Varma. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2482–2491, Osaka, Japan, 2016. The COLING 2016 Organizing Committee.

[5] Ramchandra Joshi and Raviraj Joshi. Evaluating input representation for language identification in hindi-english code mixed text. *arXiv preprint arXiv:2011.11263*, 2020.

[6] Pruthwik Mishra, Prathyusha Danda, and Pranav Dhakras. Code-mixed sentiment analysis using machine learning and neural network approaches. *arXiv preprint arXiv:1808.03299*, 2018. Submitted on 9 Aug 2018.

[7] Md Nishat Raihan, Dhiman Goswami, and Antara Mahmud. Mixed-distil-bert: Code-mixed language modeling for bangla, english, and hindi. *arXiv preprint arXiv:2309.10272*, 2023.