Emotional Classification in Hindi

Anubhav Sachan

anubhav.nits@gmail.com

February 2020

Abstract

Presently, emotional analysis is gaining importance in several research based domains to predict the emotion behind a tweet, post or an e-mail. This article describes an LSTM based deep learning classification ensemble for the analysis of various text statements in Hindi. ELMo, a deep contextualized word representation, is used for creating the embedding vectors for each of the statement which in turn is fed to the ensemble. The implementation of the model is done using PyTorch framework and the IPython Notebook along with dataset is available on Github.com.

1 Introduction

The study of identifying, classifying and analyzing emotions as expressed and reflected in a piece of text is termed as the emotional analysis. Sentiment analysis typically focuses on recognising valence − positive or negative orientation. Among the less explored sentiment areas is the recognition of types of emotions and their strength or intensity in a statement.

This work addresses the task of exploring the computational techniques for classifying the emotion in a text in the non-binary fashion. The data provided, was considerably less for a dense neural architecture, hence, a basic LSTM based deep learning ensemble is used. With the aid of softmax function in multi-class neural networks, the final prediction of an emotion for a particular statement is made.

2 Related Works

There has been extensive work in the domain of sentiment and emotion analysis. Although, there is significant intersection between techniques used for sentiment analysis and emotion analysis, yet the two are different in many ways. Emotion analysis is tackled at a fine grained level and has historically proved to be more challenging. Aman et. al. in their work *Identifying Expressions of Emotion in Text* [1] presented a novel annotation scheme which yielded the accuracy of

73.89% in the classification. *Bhaav - A Text Corpus for Emotion Analysis from Hindi Stories* [2] worked rigorously in data annotation in Hindi and classified the emotions using as basic network with an accuracy of 60% on BLSTM.

There exist various articles on internet which use the transfer learning approach to achieve high accuracy. One such article, *Emotion Detection from Hindi Text Corpus Using ULMFiT* achieves an accuracy of 90.26%. Similar to ULMFiT, FastText [3] from Facebook, Inc. also provides pre-trained classifier including with pre-trained word representations.

3 Approach

3.1 Overview

There are various available approaches for classification of emotions from text in Hindi. One of them, is using pre-trained vector embeddings to create corpus and use transfer learning with a pre-trained classifier model to achieve a high accuracy. The other corresponds to, creating everything from scratch. This work uses the pre-trained ELMo Word Representations to vectorize the sentences in data and thus fed into a LSTM based deep learning architecture which classifies the emotions based on probability distribution.

3.2 Data Pre-processing

3.2.1 Data Statistics

Emotion	**Instances**
Angry	130
Happy	151
Neutral	128
Sad	104
Total	**513**

Table 1: Split of dataset based emotions

Overall, the data contains 513 statements with the 4 emotions to be classified amongst as described in Table 1. The maximum length of a statement recorded was found to be 16 and the minimum length was 1. The mean length of a statement in the dataset provided is 7.7.

3.2.2 Word Representation and Padding

The generated text array was fed into a pre-trained ELMo [4], a deep contextualized word representation, to create a vector embedding for each of the

statements. By contextualized, the author means that instead of using a fixed embedding for each word, it (ELMo) looks at the entire sentence before assigning each word in it an embedding. Consider the case of polysemy in the sentences: *I read the book yesterday.* & *Can you read the letter now?*. The word, *read*, has different meanings in both the sentences, hence, the embeddings shall be different. On contrary, the traditional embedding algorithms like Word2Vec, etc. produce same embeddings where as, ELMo produces different embeddings for polysemous words. ELMo uses a bi-directional LSTM trained on a specific task to be able to create such embeddings.

Our pre-processor uses pre-trained ELMo word embeddings for Hindi from *elmoformanylangs*[5] developed by Harbin Institute of Technology Social Computing and Information Retrieval Research Center (HIT-SCIR). The pre-trained word representations used by our model are provided in Google Drive.

The word representations formed were of shape: (length(statement), 1024). Since, the length of the statement varies from 1 to 16, the vector representations are of varying lengths. To address such variation of length of representation vectors, we introduce the padding of a numpy array consisting of zeros to create a normalised shape of (16, 1024) for all the vectored statements. The illustration is shown in Figure 1.

```
[[ 0.13089043 -0.36257377 -0.23421884 ... -0.317504   -0.17286213
   0.25268275]
 [-0.51900053  0.10947278  0.13969344 ...  0.00311139  0.11413363
   0.15579191]
 [ 0.08198566 -0.15161669  0.26241842 ...  0.08197938 -0.47983566
  -0.42625105]
 ...
 [-0.19639985  0.09258321 -0.40550843 ...  0.27218652  0.15462351
   0.23844628]
 [ 0.15268725 -0.13790666 -0.13361748 ...  0.13847873 -0.44631806
  -0.46679655]
 [-0.06995365  0.43693706  0.29346168 ...  0.15250757 -0.12707436
  -0.17028272]]
```

(a) Sentence vector without padding

```
[[ 0.13471366 -0.35178316 -0.20498598 ... -0.32855704 -0.17134507
   0.27417138]
 [-0.57401109  0.16286895  0.15113232 ...  0.00771047  0.11669075
   0.16146442]
 [ 0.0997899  -0.1065982   0.26654178 ...  0.07326907 -0.46850815
  -0.39888182]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]
```

(b) Sentence vector with padding

Figure 1: Illustration of padding in sentence embeddings

3.3 Architecture

3.3.1 LSTM

Long Short-Term Memory (LSTM)[6] is a refined Recurrent Neural Network (RNN) based architecture having prowess to deal with long-term dependencies occurring in a time series data. Recurrent nets attempting to correlate the output on each step and final output with events several time steps before suffered a major setback since they could not figure out a way to decide how much importance the remote inputs should be given. Due to repeated matrix multiplication of the gradients coming from deeper layers, the gradients, having small values (< 1), tend to shrink exponentially till they 'vanish' or the ones with large values (> 1) tend to increase till they 'explode'. These problems were labeled as vanishing and exploding gradients respectively. LSTM overcomes this by preserving the errors that can be propagated through space and time.

A LSTM model comprises of five kinds of gate in every node i. These are denoted by five vector representations including an input gate $\overrightarrow{l_i}$, an output gate $\overrightarrow{O_i}$, a forget gate $\overrightarrow{f_i}$, a memory cell gate $\overrightarrow{c_i}$ and a candidate memory cell gate $\overrightarrow{C_i}$. $\overrightarrow{O_i}$ and $\overrightarrow{f_i}$ indicate which value should be kept, updated or forgotten. The forget gate $\overrightarrow{f_i}$ vector is for keeping the long term memory. Each node i corresponds to word vector $\overrightarrow{v^{w_i}} \in S_k$ of dimension D. The LSTM cell state c_i and hidden state h_{w_i} can be updated in two steps.

In the first step we use the previous hidden state $h_{w_{i-1}}$ (dimension of hidden layer is D) to find:

$$\overrightarrow{f_i} = sigmoid\ (\widehat{W_f} * [\overrightarrow{h}_{w_{i-1}}, \overrightarrow{V}^{w_i}] + \widehat{b_f}) \tag{1}$$

$$\overrightarrow{l_i} = sigmoid\ (\widehat{W_l} * [\overrightarrow{h}_{w_{i-1}}, \overrightarrow{V}^{w_i}] + \widehat{b_l}) \tag{2}$$

$$\overrightarrow{O_i} = sigmoid\ (\widehat{W_O} * [\overrightarrow{h}_{w_{i-1}}, \overrightarrow{V}^{w_i}] + \widehat{b_O}) \tag{3}$$

$$\overrightarrow{C_i} = tanh\ (\widehat{W_C} * [\overrightarrow{h}_{w_{i-1}}, \overrightarrow{V}^{w_i}] + \widehat{b_C}) \tag{4}$$

where $\widehat{W_f}$, $\widehat{W_l}$, $\widehat{W_O}$, and $\widehat{W_C}$ are the parameter matrix & $\widehat{b_f}$, $\widehat{b_l}$, $\widehat{b_O}$, and $\widehat{b_C}$ are the regularization parameter matrix for the respective gates, $\overrightarrow{h}_{w_{i-1}}$ is the previous hidden state vector and $\overrightarrow{V}^{w_i}$ is the word vector of the current state.

In the second step $\overrightarrow{c_i}$ is updated with the help of previous cell state $\overrightarrow{c_{i-1}}$ according to the formula given by equation 5.

$$\overrightarrow{c_i} = \overrightarrow{f}_i \odot \overrightarrow{c}_{i-1} + \overrightarrow{l}_i \odot \overrightarrow{C}_i \tag{5}$$

The current hidden state $\overrightarrow{h}_{w_i}$ can be obtained by using equation 6.

$$\overrightarrow{h}_{w_i} = \overrightarrow{O}_i \cdot tanh \overrightarrow{f}_i \odot \overrightarrow{c}_i \tag{6}$$

The last hidden $\overrightarrow{h}_{w_i}$ is taken as our final output of dimension D.

3.3.2 Our Model

Our model is an ensemble of deep learning neural networks based on LSTM (as described in 3.3.1). The input word representations of shape (16, 1024) are fed into a mono-directional LSTM network having 1 layer and hidden size 512. The output of LSTM is sent to a single classification neural layer (dense layer) with softmax function to provide the final classified emotion.

A limitation of using only LSTM architecture is that it encodes the input sequence to a fixed length internal representation. This imposes limits on the length of input sequences that can be reasonably learned and results in worse performance for very long input sequences.

To tackle such an anomaly, attention mechanism was introduced in this work. It overcomes the limitation that allows the network to learn where to pay attention in the input sequence for each item in the output sequence.

3.4 Architecture Training

To measure the effectiveness of the curated model, a thorough experimentation has been conducted and the IPython Notebook is available on Github repository for illustrating the training and validation over the set of data.

The code was run and improved upon in Google Colaboratory.

The shuffled data comprising of an array of 513 sentence embeddings was subjected to split into training and test data. The training data had 461 sentence embeddings and testing data had 52 sentence embeddings. As already mentioned, all the sentence embeddings were of shape (16, 1024). Hence, completely random data was fed into the network.

Adam's Stochastic Optimisation Algorithm[7] with a learning rate of $1 \times e^{-5}$ was used to run the training and validation (on test data) for 300 epochs. The result of such optimisation was reflected in the good cause.

4 Result

Upon rigorous experimentation, our model achieved the peak accuracy of 75% in the validation set. Corresponding to the peak accuracy on training set (250th epoch, accuracy = 66.0345%), our model achieved 73.0769% on the validation set. The figure 2 shows a comprehensive accuracy measures on training as well as the validation set over 300 epochs and learning rate = $1 \times e^{-5}$.

The model (.pth file) at 48th epoch (where peak accuracy was achieved) and at 292th epoch (the end most epoch where peak accuracy is achieved) have size of 12.6 MB each.

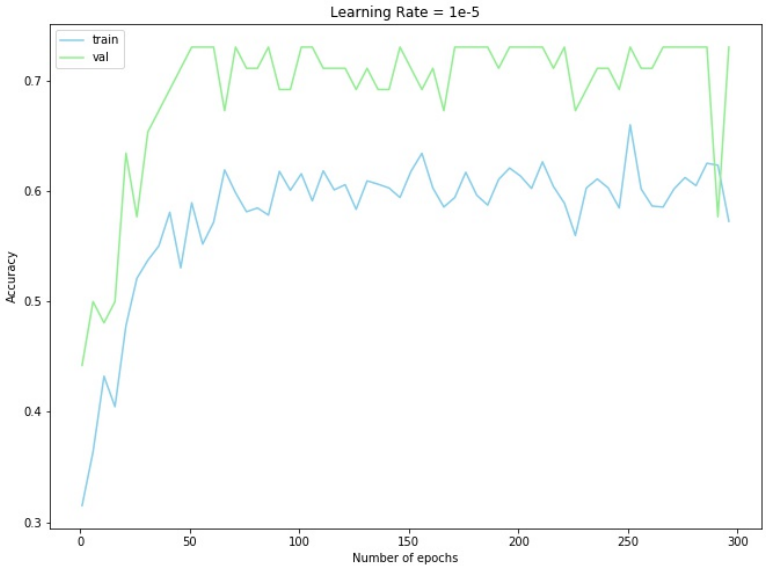

Figure 2: Accuracy of training and validation set over number of epochs

5 Challenges

5.1 Language Specific

The computational methods used in the tasks pertaining to sentiment analysis can readily be applied to the emotion analysis tasks. Therefore, the challenges for emotional analysis from text are very similar to that of the domain of sentimental analysis from text.

5.1.1 Word Order

The order in which words appear in a sentence plays an important role in determining polarity as well as subjectivity of the text in Hindi. Hence, while creating embeddings, ELMo is used. The major reason for selecting ELMo pretrained representations over Word2vec is the fact that the latter takes input words and output word embeddings where as ELMo, in contrast, is a character based model using character convolutions and can handle out of vocabulary words.

5.1.2 Handling Spelling Variations

A word with the same meaning can appear with multiple spelling variations. Occurrence of such variations can pose challenges for the machine learning models that has to take into account all the spelling variants.

5.1.3 Lack of Resources

The lack of lexicons, developed techniques and elaborate resources in Hindi also adds to the challenge. Even, the embedding used, *elmoformanylangs*, contains word embedding size of 220,391 and character word embedding size of 4,433.

6 Conclusion

Due to the lack of availability of dataset in Hindi, the pre-trained word embeddings and a deep learning model built from scratch is used which provides satisfactory accuracy of 75%. The accuracy for the given dataset can be improved using transfer learning and a little modification in the architecture.

References

[1] S. Aman and S. Szpakowicz, "Identifying expressions of emotion in text," in *International Conference on Text, Speech and Dialogue*, pp. 196–205, Springer, 2007.

[2] Y. Kumar, D. Mahata, S. Aggarwal, A. Chugh, R. Maheshwari, and R. R. Shah, "Bhaav- a text corpus for emotion analysis from hindi stories," 2019.

[3] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *CoRR*, vol. abs/1612.03651, 2016.

[4] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *CoRR*, vol. abs/1802.05365, 2018.

[5] W. Che, Y. Liu, Y. Wang, B. Zheng, and T. Liu, "Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, (Brussels, Belgium), pp. 55–64, Association for Computational Linguistics, October 2018.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.