CMPUT 501 Project Report : Sentiment Analysis of Code-Mixed Social Media Text (SemEval 2020 Task 9)

Arnob Mallik and **Arif Hasnat**
Department of Computing Science
University of Alberta

Abstract

Sentiment analysis is the task of predicting the polarity of a sentence. In this work, we present a method of predicting the sentiments of Hindi-English code-mixed tweets. Conventional methods of sentiment analysis do not work well in case of code-mixed texts as the structure of traditional English sentences are quite different from these ones. We use Google's Universal Sentence Encoder to generate sentence embeddings and also extract additional features from various Hindi and English lexical resources. These features and embeddings are fed to a Deep Neural Network which predicts the sentiment of the given tweet.

1 Introduction

Code-mixing is the mixing of two or more languages or language varieties in speech.[1] Non-English speakers, specially Indians often interchange English words and Hindi words written in Roman scripts to express their opinions on social media platforms such as Twitter and Facebook. The Goal of this project is to predict the sentiment of such code-mixed tweets.[2] The input will be English-Hindi code-mixed tweet and the output will be one of the sentiment labels-positive, negative, or neutral.

The evolution of social media texts such as blogs, micro-blogs (e.g., Twitter), and chats (e.g., WhatsApp and Facebook messages) has created many new opportunities for information access and natural language research. The one great advantage of social media is that, there is so much data to gather from so many people. From a business perspective, companies can get an idea about how people feel about their product/service by doing sentiment analysis on data crawled from social media. This also allows companies to capitalize on positive sentiment trends and address criticisms in a more proactive manner. Sentiment analysis can also be used to track political trends and predict results of elections.

However, code mixing i.e. interchanging of multiple languages has posed new challenges to researchers concerned with sentiment analysis. For example, lets consider the tweet "*@kunalkamra88 @Swamy39 Tere maaliko ko jail yahi bhejenge.. Fir payment k lie rote rehna"*. Here "jail" and "payment" are English words and others are Hindi words. These type of texts differ significantly from traditional English texts and different approaches are required to process them efficiently.

In this study, to predict sentiments of code-mixed tweets, we propose a deep neural network based approach which uses Universal Sentence Encoder (Cer et al., 2018) for emebeddings and several lexical features as additional input. Our approach outperforms MVC and Naive Bayes baselines on the SemEval dataset [2] and achieves similar results as Prabhu et al. (2016) when evaluated on their own dataset.

2 Related Work

Sentiment analysis of social media text has received a lot of interest from the research community in the recent years with the rise to prominence of Facebook and Twitter. Prabhu et al.(2016) introduced a Sub-Word Long Short Term Memory model to learn sentiments in a Hindi-English Code Mixed dataset. They also introduced a dataset of their own, on which we trained our model and compared the results.

Zhang et al.(2011) suggested combining learning-based and lexicon-based techniques using a centroid classifier. In our approach, we do the same as we take learning based features from the Universal Sentence Encoder (Cer et al., 2018) and extract lexicon-based features from English SentiWordNet [3] and Hindi SentiWodNet (Das and Bandyopadhyay, 2010)

Ghosh et al.(2017) used extensive pre-processing to remove noise from raw text. Then they used multilayer Perceptron model to determine the polarity of the code-mixed social media text. Some of our pre-

[1] https://en.wikipedia.org/wiki/Code-mixing
[2] https://competitions.codalab.org/competitions/20654
[3] http://sentiwordnet.isti.cnr.it/

processing and feature extraction methods are similar to them as both works are on code-mixed tweets.

3 Methodology

Algorithm 1 Training

procedure TRAIN($train\_data$)
  $train\_X = train\_data[0]$
  $train\_Y = train\_data[1]$
  $Feature\_X = []$
  $Processed\_X = PreProcess(train\_X)$
  $Embeddings = Sent\_Embedding(Processed\_X)$
  $Other\_Features = Feature\_Extraction(Processed\_X)$
  $Feature\_X.append(Embeddings)$
  $Feature\_X.append(Other\_Features)$
  $classifier = DNN\_classifier$
  $classifier.fit(Feature\_X, train\_Y)$
  return $classifier$

Algorithm 2 Testing

procedure TEST($test\_data, classifier$)
  $test\_X = test\_data[0]$
  $test\_Y = test\_data[1]$
  $Feature\_X = []$
  $Processed\_X = PreProcess(test\_X)$
  $Embeddings = Sent\_Embedding(Processed\_X)$
  $Other\_Features = Feature\_Extraction(Processed\_X)$
  $Feature\_X.append(Embeddings)$
  $Feature\_X.append(Other\_Features)$
  $predictions = classifier.predict(Feature\_X)$
  $evaluate(test\_Y, predictions)$
  return $predictions$

3.1 Data Preprocessing

We performed the following steps to preprocess the raw data:

- Replaced "https://...." with the string "URL".
- Replaced "@name" with the string "USER".
- Replaced "#word" with the string "HASHTAG"
- Removed all special characters and punctuations.
- Replaced multiple spaces with one.
- Remove all single character words.
- It is often found in social media text that certain characters are repeated more than once (Ghosh et al., 2017). For example, lol (abbreviated form of laughing out loud)can be written as loool, looool or looooooool. We tackle this issue by replacing characters that occur more that two times in a row with two instances of the same character.

3.2 Sentence Embeddings

In case of text data, count based models such as Bag of Words deal with individual words, which may have their own identifiers and do not capture semantic relationship among words. This leads to sparse word vectors for textual data. On the other hand, if we have a numeric representation of text data which captures even the context and semantics, then we can feed it to deep learning models for our task.

While word embeddings can produce representations for words which can capture the linguistic properties and the semantics of the words, recent work has demonstrated strong performance using pre-trained sentence level embeddings (Conneau et al., 2017). For our task, we are going to use the Universal Sentence Encoder (Cer et al., 2018) which converts text into high-dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks.

This model is trained on a variety of data sources with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length text and the output is 512 dimensional vector. These vectors could now be used as features in any modelling techniques.

3.3 Language Identification

For some of our features, we had to identify the language of the words of the test data. We used the NLTK Corpus [4] to detect the English words. The words that were not found in the corpus were considered as Hindi

3.4 Feature Extraction

Aside from the sentence embedding vectors, we generated the following features from the text data:

- **English Scores** : We used SentiWordNet (SWN)[3] as a lexical resource in our work. SWN is a resource which assigns positive, negative and objective scores to English words. For each sentence, we sum up these scores for all the English words which are present in the SWN list. Thus we get three distinct features namely *English_Positive_Score* *English_Negative_Score*

[4] https://www.nltk.org/api/nltk.corpus.html

and *English_Objective_Score*. Finally, we normalized these scores.

- **Hindi Scores** : Similarly for Hindi words we use the Hindi SentiWordNet (Das and Bandyopadhyay, 2010). However, the Hindi words in this SentiWordNet are written in Devnagari (Hindi) script, while the Hindi words in our data are written in Roman (English) script. Hence we use the indic-transliteration 1.8.9 library [5] to convert our Hindi words into the Devnagari script. Then we can use the Hindi SentiWordNet (Das and Bandyopadhyay, 2010) which assigns positive, negative and neutral scores to Hindi words. For each sentence, we sum up these scores for all the Hindi words that are present in the Hindi SentiWordNet list. Thus we get three distinct features namely *Hindi_Positive_Score*, *Hindi_Negative_Score* and *Hindi_Neutral_Score*. Finally, we normalized these scores.

- **Profanity** : We used the Hinglish profanity dataset built by Mathur et al. (2018) to detect profanity in our sentences. The dataset contains scores for each offensive word. For each sentence, we sum up the profanity scores for all offensive words and thus we get the *Profanity* feature.

These features along with the sentence embeddings are fed to the classifier.

3.5 Classifier

We used a Deep Neural Network to classify the code-mixed tweets. After the tuning process using 3-fold cross validation, we settled on the following parameters that produce the best results:

- Two hidden layers of 512 and 128 nodes respectively.
- AdaGrad Optimizer
- Learning rate = 0.005
- Batch size = 512

4 Evaluation

4.1 Datasets

In our dataset [2], we have 15131 train examples and 1869 test instances. The distributions of train and test data are shown in table 1 and 2 respectively.

[5] https://pypi.org/project/indic-transliteration/

Class	Instances
Positive	5034
Negative	4459
Neutral	5638

Table 1: Train Data

Class	Instances
Positive	582
Negative	533
Neutral	754

Table 2: Test Data

4.2 Evaluation Metrics

We calculated Precision, Recall, F1 Score and Accuracy to evaluate our results.

F1 Score is calculated as:

$$F1\_Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

We also calculated Weighted F1 score by weighting the F1 score of each class by the number of samples from that class.

4.3 Results

We have got 56.23% accuracy for predictions on test data and the results are summarized in table 3.

	Positive	Negative	Neutral
Positive	352	66	164
Negative	51	316	166
Neutral	207	164	383

Table 3: Confusion Matrix

From 3, we can calculate the precision and recall for each class and the the scores are shown in table 4.

Class	Precision	Recall	F1 Score
Positive	60.48%	57.70%	59.57%
Negative	59.29%	57.88%	58.58%
Neutral	50.80%	53.72%	52.22%

Table 4: Precision, Recall and F1 Scores

$$Weighted\_F1\_Score = \frac{105266.76}{1869} = 56.32\%$$

4.4 Comparisons

4.4.1 With Baselines

- **MVC Baseline** : By assigning the most frequent label seen in the training set (neutral) to all test instances, we get an accuracy of 40.5% on test data, whereas our model gets an accuracy of 56.23%.

- **Naive Bayes** : By running the Naive Bayes algorithm on the train and test data, we achieved an accuracy of 52%, whereas our algorithm achieves an accuracy that is 4.23% better.

4.4.2 With Alternative Approaches

Since this is the first iteration of the Code-Mixed SemEval task, we could not compare our approach to other works that uses the SemEval dataset.[2]

Hence, we trained and tested our model using the dataset proposed by Prabhhu et al. (2016) and compared our results with theirs. There were 3878 total examples in their dataset and among them 1353 were positive, 569 were negative and 1957 were neutral. We divided the data into randomized 80-20 train-test split as suggested by Prabhu et al. (2016). We got an accuracy of 69.12%, which is very similar to theirs (69.7%).

4.5 Error Analysis

We have identified the following points as the main reasons behind the wrong predictions by our model:

- **The Neutral Class** From table 3, we can see that the number of positive examples classified as negative and vice-versa are quite low (51 and 66) respectively. However, the misclassifications involving the neutral class are much higher. This indicates that we were not able to generate features that properly distinguish the neutral examples from the positive and negative ones.

- **Lexical Resource** We have used the Hindi SentiWordNet (Das and Bandyopadhyay, 2010) to extract Hindi lexical scores. However, out of 15131 sentences in the train data, we had only about 1000 words that exist in the wordnet. This shows that, there is a need for a richer lexical resource that contains non-standard words typically used in social media.

- **Transliteration** We used the indic-transliteration library[5] to convert Hindi words written in Roman script to original (Devnagari) script. However, one Hindi word can be written in various ways using Roman script. For example, the word "pyaar" (meaning : love) can be written as "peyar", "piyar", "pyr", "piyaar" etc. The transliteration of all these words should result in the same Hindi word. Our indic-transliteration library fails in these cases and hence there is a need for a more sophisticated method for transliteration.

- **Bias** Finally, we think the bias in the dataset also impacts the results. For example, the tweet "*@idevadhikari Dev da tmr new movie song Ami tmk valobasi.really loved ???.lot ????????*" is originally tagged as "neutral". This tweet is about a person loving a movie song and one may argue that it should be labelled as "positive".

5 Conclusion

In this work, we presented a deep learning approach for predicting the sentiments of Hindi-English code-mixed tweets. We encoded the tweets using Google's Universal Sentence Encoder and also extracted English and Hindi lexical features from various external resources. These features along with the encodings were used as input to a DNN classifier. The results show that our approach outperforms the MVC and Naive Bayes baselines. There is also much room for improvement and in future we will try to find/build better lexical resources for this task and will also try to extract distinguishing features specially with a focus on the neutral class.

Acknowledgments

The authors contributions are as follows:

Arnob Mallik	Arif Hasnat
Literature Review	Preprocessing
Sentence Embeddings	Language Identification
Exploring other Features	Hindi Feature Extraction
Exploring Classifiers	English Feature Extraction
Writing Methodology	Code Management
Writing Evaluation	Writing Related Work
Writing Conclusion	Writing Introduction

References

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* .

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364* .

Amitava Das and Sivaji Bandyopadhyay. 2010. Sentiwordnet for indian languages. In *Proceedings of the Eighth Workshop on Asian Language Resouces*. pages 56–63.

Souvick Ghosh, Satanu Ghosh, and Dipankar Das. 2017. Sentiment identification in code-mixed social media text. *arXiv preprint arXiv:1707.01184* .

Puneet Mathur, Ramit Sawhney, Meghna Ayyar, and Rajiv Shah. 2018. Did you offend me? classification of offensive tweets in Hinglish language. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*. Association for Computational Linguistics, Brussels, Belgium, pages 138–148. https://doi.org/10.18653/v1/W18-5118.

Ameya Prabhu, Aditya Joshi, Manish Shrivastava, and Vasudeva Varma. 2016. Towards sub-word level compositions for sentiment analysis of hindi-english code mixed text. *arXiv preprint arXiv:1611.00472* .

Lei Zhang, Riddhiman Ghosh, Mohamed Dekhil, Meichun Hsu, and Bing Liu. 2011. Combining lexicon-based and learning-based methods for twitter sentiment analysis .