"Did you really mean what you said?" : Sarcasm Detection in Hindi-English Code-Mixed Data using Bilingual Word Embeddings

Akshita Aggarwal, Anshul Wadhawan, Anshima Chaudhary and Kavita Maurya
Department of Computer Engineering
Netaji Subhas University of Technology
Dwarka, New Delhi
{akshitaa, anshulw, anshimac, kavitam}.co.16@nsit.net.in

Abstract

With the increased use of social media platforms by people across the world, many new interesting NLP problems have come into existence. One such being the detection of sarcasm in the social media texts. We present a corpus of tweets for training custom word embeddings and a Hinglish dataset labelled for sarcasm detection. We propose a deep learning based approach to address the issue of sarcasm detection in Hindi-English code mixed tweets using bilingual word embeddings derived from FastText and Word2Vec approaches. We experimented with various deep learning models, including CNNs, LSTMs, Bi-directional LSTMs (with and without attention). We were able to outperform all state-of-the-art performances with our deep learning models, with attention based Bi-directional LSTMs giving the best performance exhibiting an accuracy of 78.49%.

1 Introduction

With the advent of social media, a large part of human interaction is carried out online. This leads to generation of huge amounts of textual data that can be used to draw meaningful inferences. Social media websites like Facebook, Twitter, Reddit etc are used by people across cultures to communicate with each other and voice their opinions.

With the large amount of data available from social media, the study of various types of linguistic expressions like irony, humor, sarcasm, aggression, hate etc has become a keen research area. Especially in the field of NLP, automatic detection of these expressions is being widely explored (Joshi et al., 2017). Automatic detection involves using computational methods to detect the presence of a particular emotion.

Although English is the language most commonly used on these websites, a majority of people are not native English speakers. These people therefore prefer to communicate in languages other than English (Danet and Herring, 2007). A study on the languages that are most commonly used for exchange of information on Twitter showed that around 50% of the posts are written in languages other than English (Hong et al., 2011). This raises the opportunity of dealing with multi-lingual data generated by the social media sites. Various statistics show that around 26% of the Indian population is bilingual[1]. This gives rise to the phenomenon of code-switching and code-mixing (Gupta et al., 2016). Code mixing takes place when speakers use two or more languages below clause level in a single social context. Multilinguals use such a mixture of languages, particularly on social media (Mónica et al., 2009). There are multiple challenges of working with code-mixed data like large amount of new constructions that are a result of combining lexicons and syntax of two different languages, availability of very small amounts of annotated data and use of very different approaches when compared to mono-lingual data (Çetinoğlu et al., 2016).

In this paper, we wish to work on detecting one of the most popular linguistic constructs used across social medias, 'sarcasm'. The cambridge-dictionary[2] defines sarcasm as 'the use of remarks that clearly mean the opposite of what they say'. Example: "You have been working hard," he said with heavy sarcasm, as he looked at the empty page.

Starting with the earliest known work which focuses on sarcasm detection in speech (Tepperman et al., 2006), this domain has been widely explored in sentiment analysis. Since sarcasm is a sentiment, detection of sarcasm is important in order to predict

[1] https://en.wikipedia.org/wiki/Multilingualism_in_India
[2] https://dictionary.cambridge.org/

Proceedings of the 2020 EMNLP Workshop W-NUT: The Sixth Workshop on Noisy User-generated Text, pages 7–15

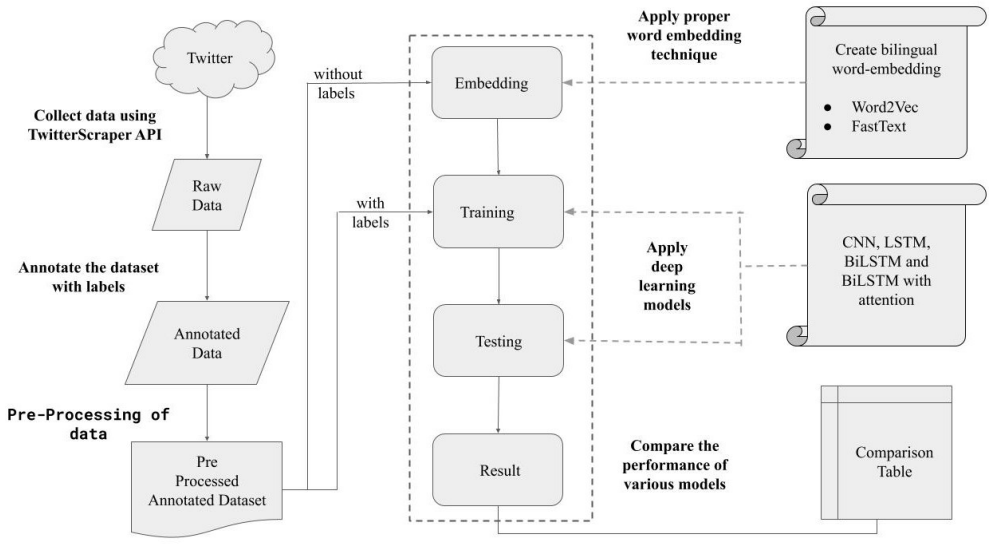

Figure 1: Proposed Methodology

the sentiment of a sentence. Being a challenging problem, automatic detection of sarcasm has been a popular area of research.

Although a lot of work has been carried out on sarcasm detection in English (Davidov et al., 2010; Bamman and Smith, 2015), the detection of sarcasm in code-mixed language like Hinglish (Hindi-English) is relatively unexplored. The current state-of-art performance is proposed via a random forest model on a dataset of 5000 Hinglish tweets (Swami et al., 2018).

The contributions of our work includes :

1. In this paper, we have experimented with deep-learning approaches to detect sarcasm in the Hindi-English code mixed dataset. The corpus prepared is released along with the paper.

2. Deep learning is being used extensively in the domain of natural language processing and has given satisfactory results (Young et al., 2018). In our work, we propose five different deep learning models namely, Series CNN, Parallel CNN, LSTM, Bi-directional LSTM and Bi-directional LSTM with attention.

3. The proposed models take self-trained bilingual word embeddings generated by Hindi-English code mixed data as input.

4. Our work present an alternate approach to the work done using traditional machine learning models like SVMs and random forests (Swami et al., 2018).

2 Proposed Methodology

2.1 Dataset Creation

The current dataset provided in paper (Swami et al., 2018) contains 5250 tweets, out of which 504 tweets are labelled as sarcastic while the remaining 4746 tweets are labelled as not sarcastic. All the deep learning models seemed to erroneously predict all tweets to be not sarcastic, since this dataset is highly skewed as well as insufficient. Therefore, to meet the model needs, we created a larger class-balanced dataset by scraping relevant tweets from twitter using TwitterScraper API[3] with search tags like #sarcasm, #irony, #humor, #bollywood, #cricket along with some common hindi words to obtain Hinglish data.

2.2 Dataset Annotation and Analysis

We were able to obtain around 427k tweets for training the proposed deep learning models. After carefully filtering out the obtained tweets for Hindi-English code mixed entries, we were successful in creating a corpus of 100k Hindi-English code mixed tweets with 49% entries being sarcastic and remaining 51% being non-sarcastic. The annotation scheme was based on the search tags(hashtags) used for scraping the tweets. We marked all examples fetched with hashtags like sarcasm, irony etc to have a positive sarcasm label, whereas all examples with generic hashtags like cricket, bollywood etc to have a negative sarcasm label. This

[3] https://github.com/taspinar/twitterscraper

Category	Tweet Count
Total Tweets	106899
Sarcastic	52587
Non-Sarcastic	54312

Table 1: Tweets per category

annotation scheme was susceptible to noise, however, as a quality check measure, we manually traversed the data and noticed that the noisy examples were meagre in proportion. Also, the noisy examples were necessary for the models to generalize well on the diverse dataset we obtained. Having a class-balanced dataset was significant to our problem to ensure that deep-learning models learn the right trends, not being biased towards a particular class. Embeddings were initally trained on solely Hinglish data, which was later on added with English data. The embedding training dataset, labelled sarcasm detection dataset and the proposed deep learning classification models are made available online [4] to facilitate further research.

Examples of some annotated data :

Tweet: Koi Rah Mushkil Nahi hain bus vo rah #bengalurutraffic se bach jaayein #sarcasm @random

Translation No path is difficult as long as it does not pass through Bangalore traffic. (Bangalore is an Indian city infamous for it's traffic)
Sarcasm : YES

Tweet : Hindustan ke tamam log chahte h ke jis trah se auraton ke upar crime bhadr rha h gang rape ke waqia ho rha iske liye central govt wali modi sarkar 1 strong law bnaye

Translation: All indians want Modi government to make strong laws on crime against women
Sarcasm: NO

2.3 Data Preprocessing

The data obtained from social media is very noisy and a lot of preprocessing is required. While creating the dataset, we removed the '#' symbols from the data, along with removing all the mentions (@). We also removed rare words (words having occurrence of less than 10 in the entire dataset) and search tags (like cricket, sarcasm) to avoid our deep learning models being biased towards certain words while learning. Further, URLs and punctuation marks were also removed.

2.4 Creation of Hindi-English Bi-lingual Word Embeddings

Being a text classification problem, it is essential for the words of the dataset to be first converted to vector representations. Word embedding is learned from unannotated plain text, useful in determining the context in which a given word is used. They provide a dense vector representation of syntactic or semantic aspects of a word (Mandelbaum and Shalev, 2016). To create a Hindi-English word embedding, we needed a huge amount of data. We used TwitterScraper API to extract 427k Hinglish tweets and 300k English tweets from Twitter for Hindi-English code-mixed data. For the Hinglish code mixed tweets, we removed the tweets obtained in pure Devnagri and kept only those which were a mixture of both Hindi and English sentences. The above obtained dataset was further processed to remove rare words, hashtags and mentions to obtain a less noisy corpus for training word embeddings.

We experimented with 2 different kinds of word embeddings for two types of datasets, one which solely consisted of Hinglish tweets, the other which consisted of 300k English along with Hinglish tweets. We chose to experiment with a mixture of Hinglish and English tweets in order to get the co-relations between the words of the two languages. Each of these variations, after similar processing (removing hashtags, URLs, punctuations, user mentions and keywords used for scraping), were tried for two types of embeddings:

Word2Vec: In this embedding, words in the corpus are converted into vectors, where words that share common context are placed closed to each other in the vector-space (Mikolov et al., 2013). Since Word2Vec is pre-trained for English dataset only, we had to train our model on custom Hindi-English code mixed dataset, to obtain Hinglish word embeddings.

FastText: FastText which was given by Facebook in 2016, is an addition to the Word2Vec embeddings (Joulin et al., 2017). Rather than giving individual words to a model, FastText breaks down the words into multiple sub-words, also known as n-grams (Bojanowski et al., 2017). During the training of the model, weights are learned for all

[4] https://github.com/Akshitaag/Sarcasm_Detection

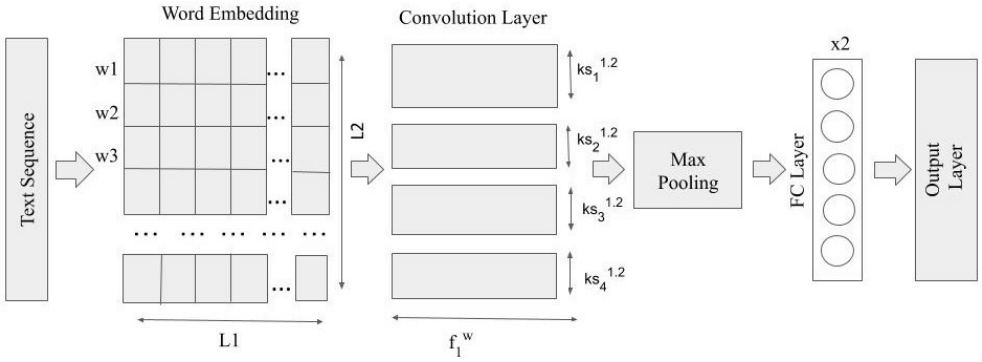

Figure 2: CNN Model 1.2 architecture

the n-grams along with the complete word. Unlike Word2vec, rare words can be appropriately featured as now it is much more likely that some of their n-grams also occur in other words. This is especially true for social media text where people use multiple spellings for the same words (amaze, amazeee, amazing, amazinggg).

2.5 Deep Learning Models

We propose 5 different models to experiment with the above problem. The models tested include Series CNN, Parallel CNN, LSTM, Bi-directional LSTM and Bi-directional LSTM with attention. Word embeddings, as generated by FastText and Word2Vec custom data trained representations, served as input to the models, generate as output a binary variable depicting the probability of the corresponding tweet being sarcastic.

2.5.1 Convolutional Neural Networks (CNN)

CNNs have the ability to extract features from data provided to it as input. In our case, the input data is a set of word vectors, over which convolution operation is performed to extract features and thereby, perform classification.

We propose 2 CNN deep learning model architectures, one which has convolution layers in series, denoted by model 1.1 and the other which has convolutional layers in parallel, denoted by model 1.2. Both the models have an embedding layer as the first one, which is used to select the word vector representations corresponding to the words of the tweet under consideration during the training session, from the word embedding matrix. In model 1.1, the embedding layer is followed by a couple sets of convolution and max pooling layers in series whereas in model 1.2, it is followed by 4 single dimensional convolution layers in parallel.

The convolution layer is responsible for the extraction of features from the word vectors provided as input. The outputs of these layers, concatenated in case of model 1.2, are fed to a global max pooling layer with a dropout activated. This layer is further followed by 3 dense fully connected layers, the final layer with a single neuron, which is responsible for the classification. We have used dropout in the the global max pooling layer so as to reduce overfitting, which already is low due to the large dataset. However, on its application, the difference between the validation accuracy and training accuracy reduced, also leading to better convergence.

2.5.2 Recurrent Neural Networks (RNN)

The meaning of a word depends on the context in which it is used. For example,

Sentence 1 : We dined at a small Mexican restaurant and spent the meal discussing general topics.
Sentence 2 : General Zod is an enemy of Superman.

The word general, in the above sentences, carries different meaning depending on the context in which it is used. Thus, in order to record the context of a particular word, i.e. the words surrounding the word under consideration, RNNs are used. There are different ways to capture the context of a particular word, each having its unique mechanism to model the meaning of the word depending on words coming before and after the word.

The RNN model equations and corresponding notation have been taken from (Yu et al., 2015). Given an input sequence $x = (x_1, x_2, ..., x_{t-1}, x_T)$, the output vector sequence $y = (y_1, y_2, ..., y_{T-1}, y_T)$ and hidden vector sequence $h = (h_1, h_2, ..., h_{T-1}, h_T)$ are computed in a standard recurrent neural

network by evaluating the below equations from t = 1 to t = T:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_0$$

where weight matrices are denoted by W terms, bias vectors are denoted by b terms, and hidden layer function is given by $\mathcal{H}$.

Long Short-Term Memory (LSTM): LSTMs have been successfully applied to binary text classification problems like political text classification (Rao and Spasojevic, 2016), by capturing the appropriate context. Also, the vanishing gradient problem in RNNs has been addressed successfully by LSTMs (Hochreiter and Schmidhuber, 1997). The context of a word depends on the words occurring before the word under consideration. In order to model this scenario, an LSTM based network is constructed.The LSTM design comprises of a set of repetitively associated subnets, known as memory blocks. Each block contains at least one self-associated memory cells along with three multiplicative units - the input, output and forget gates - that give regular functionality of write, read and reset operations to the cells.

A LSTM network is framed precisely like a basic RNN, other than the nonlinear units in the hidden layers being supplanted by memory blocks. The multiplicative gates permit LSTM memory cells to store and access data over extensive stretches of time, in this manner maintaining a strategic distance from the vanishing gradient issue. For instance, as long as the input gate stays shut (has an activation near 0), the activation of the cell won't be overwritten by the new inputs showing up in the network, and can in this manner be made accessible to the net a lot later in the succession, by opening the output gate. This allows the LSTM network to carry forward semantic qualities of initial parts of the sentence to the later parts. In our architecture, an LSTM layer is appended to the embedding layer in turn followed by 2 dense fully connected layers. The final output layer has a single neuron carrying out the classification depending on the extracted context based features. LSTM blocks have the structure as shown in Figure 3, and are based on the equations presented below :

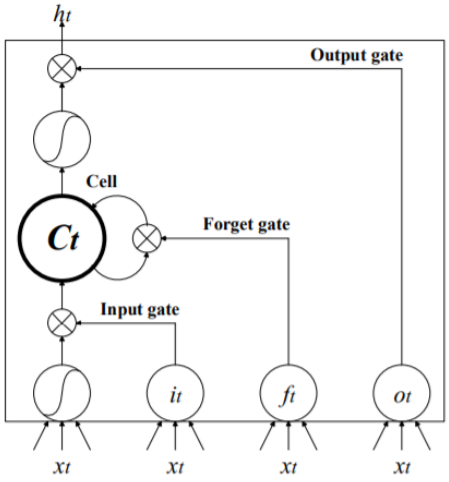

Figure 3: LSTM block structure

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$

$$h_t = o_t \tanh\left(c_t\right)$$

where logistic sigmoid function is denoted by σ, the input gate, forget gate, output gate, and cell activation vectors are denoted by i, f, o and c, all having the same size as the hidden vector h. The hidden-input gate matrix is represented by W_{hi}, and the input-output gate matrix is represented by W_{xo}.

Bi- directional LSTM: Bi-directional LSTMs have been applied and proved to be successful in capturing the context for text classification tasks (Wang et al., 2016). The context of a word not only depends on the words occurring before it, but also on the words occurring after it. Modelling this requires memory cells in the backward direction which maintain the history of words along with cells in the forward direction for the words not yet explored. To achieve this and capture the composition semantics of Hindi-English code mixed data, two LSTM layers are appended to the input embedding layer. The output features from the two layers, after concatenation ($\overrightarrow{h_t}$, $\overleftarrow{h_1}$), are flattened and fed to 2 dense fully connected layers. The classification is performed by a single neuron, as in all other models. The BiLSTM computes the forward hidden sequence $\overrightarrow{h_t}$ by traversing the

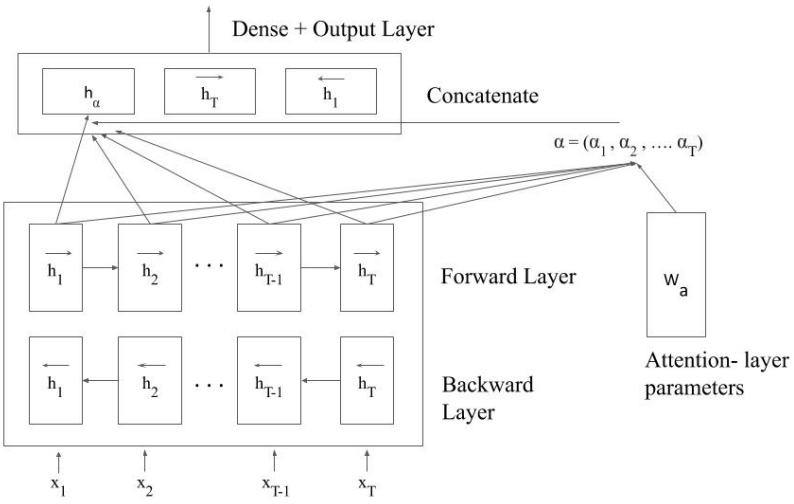

Figure 4: Attention based Bi-directional LSTM model architecture

forward layer from t = 1 to T, the backward hidden sequence $\overleftarrow{h_t}$ by traversing the backward layer from t = T to 1, and updates the output y_t as :

$$\overrightarrow{h_t} = \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\vec{h}_{t+1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h_t} = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y$$

Attention based Bi-directional LSTM: Here, we propose a technique based on attention along with the bi-directional LSTM network. The attention based network focuses on filtering the noisy elements of a sentence by learning the words which cause the greatest effect towards deciding the final output (sarcastic or not sarcastic) of the sentence under consideration. While the bi-directional LSTM network uses concatenated ($\overrightarrow{h_T}$, $\overleftarrow{h_1}$) which is then fed to the dense layers, attention based bi-directional LSTM network is different in the concatenation process. Along with the above state representations, ($\overrightarrow{h_T}$ denoting the final state representation in the forward direction and $\overleftarrow{h_1}$ denoting the first state representation in the backward direction), the attention based network inculcates the weighted summation, calculated by detecting the influence of each word, of all the time steps (denoted by $\overrightarrow{h_t}$, $\overleftarrow{h_t}$). Thus, all these hidden states are concatenated and passed on to 2 dense fully connected layers. Final classification is performed by a single neuron, as usual.

3 Experimental Settings

For the training sessions, we made a ten percent validation split and shuffled the training dataset, so that the model does not capture sequence trends, if any, in the training data, for a total of 20 epochs. The model checkpoints were saved at every epoch and those checkpoints which were saved before the model begins to overfit and the difference between the training and validation accuracies becomes significant, were used to calculate the accuracy numbers on the ten percent test dataset split. There are many important hyper-parameters in the training script of embeddings as well as the proposed models, which are tuned to produce the best training results on the validation data split. For training the word embeddings (both Word2Vec and FastText), we used an embedding size of 300, window length of 10 and negative sampling polarity. In all the models, adam optimizer along with binary cross entropy loss function has been used. All the layers have relu activation function with the exception of output layer having sigmoid activation function. We evaluated the performance of CNN models with different values for kernel_size, number_of_kernels, dropouts and strides. The best results are obtained with the following values :

stride = 1, number_of_kernels = 200, dropout = 0.5
$ks_1^{1.2}$ = 3, $ks_2^{1.2}$ = 6, $ks_3^{1.2}$ = 9,
$ks_4^{1.2}$ = 12, $ks_1^{1.1}$ = 7

For all the proposed RNNs, the following

Traditional Models	Accuracy
Naive Bayes	54.17
Random Forest	63.37
Linear SVM	69.04
RBF Kernel SVM	71.23

Table 2: Accuracy of ML models

DL Models	Hinglish Data		Hinglish + English Data	
	Word2Vec (a)	FastText (b)	Word2Vec (a)	FastText (b)
1.1 Series CNN	72.86	72.65	74.09	73.51
1.2 Parallel CNN	74.28	73.41	75.00	74.32
2.1 LSTM	76.19	75.25	77.24	75.55
2.2 Bi-LSTM	77.12	76.25	78.28	77.12
2.3 Attention Bi-LSTM	78.19	77.11	78.40	78.06

Table 3: Accuracy of DL Models

hyper parameter combination is used :

dropout_for_recurrent_state = 0.2,
dropout_for_input_state = 0.2,
number_of_LSTM_units = 150

We used the same hyper parameter values for models 2.2 and 2.3, as in model 2.1, so as to study the impact of imposing bidirectional nature to the LSTM layer, as well as exploring the effect of attention introduction. The parameters resulted in best outputs as confirmed later by trying out different values for the same.

4 Results

The dataset, as presented in (Swami et al., 2018), being insufficient and skewed for our deep learning model architectures, we ran the state-of-the-art models on our proposed dataset to carry out unbiased accuracy comparison of state-of-the-art techniques and neural network based models.

The results for the same have been presented in Table 2. Using all features, the traditional state-of-the-art models: RBF kernel SVM, random forest and linear SVM, proposed the best accuracy of 71.23% on the proposed corpus. We tested all the deep learning models with both Word2Vec and FastText based word representations. The results of both have been presented in Table 3 where model (a) and (b) refer to application of Word2Vec and FastText generated word embeddings respectively.

To the best of our knowledge, we are the first to implement and analyze deep learning model architectures and different word representations for detection of sarcasm in Hindi-English code-mixed data with a dataset large enough for deep learning models. All the proposed deep learning models performed better than the traditional state-of-the-art models, where the attention based Bi-directional LSTM network produced the best accuracy of 78.49%. In Table 3, we present the results of our proposed deep learning models for both Word2Vec and FastText based word representations, differing in the type of datatsets being used to produce the word embeddings. Overall accuracies of all models are greater when embeddings trained on Hinglish plus English data, rather than just Hinglish data are used. One possible reason for this observation can be the additional coverage of semantics and coorelations between the word vectors of English data, which can be used for code mixed Hinglish data, thus providing additional knowledge and serving as prior information for Hinglish embeddings data. The process works analogous to a knowledge transfer step in which embeddings for English data are used as prior knowledge for embeddings of Hinglish data. Moreover, Word2Vec embeddings produce better results than FastText embeddings, for all the models. One major reason for this observation is the presence of code mixed data which does not allow character n-grams to be the primary criteria for classification, in the case of FastText embeddings, since the character n-grams belong to the constructs of two different languages. Due

to the same reason, context based word vectors i.e. the Word2Vec representations perform better than the character n-grams representations in case of FastText embeddings.

The lack of clean data and linguistic complexities associated with code-mixed data are the major challenges related to the task of sarcasm detection in Hindi-English code mixed data. To allow the model to accommodate the noise in textual data, spelling errors, multiple contexts, and stemming words, even larger data is required along with cautiously labelled classes.

5 Conclusion

Social media, in recent years, has become a medium widely used by people for expression of thoughts and opinions, further leading to the realisation of tasks like emotion analysis and opinion mining. Sarcastic content in these texts make it even more challenging to figure out the overall sentiment of the text, thus needing proper processing and analysis.

In this paper, we presented a class-balanced Hindi-English code mixed dataset for the problem of sarcasm detection, by scraping relevant tweets from twitter. We compared two representations, FastText and Word2Vec, both based on different word representation learning mechanisms and trained on custom scraped data from scratch. We created two versions of embeddings, one trained with purely Hinglish data, the other with a mixture of Hinglish and English data, and compared the performance in each case. We analyzed the performance of different deep learning models, which take as input the generated word embeddings, to solve the problem of sarcasm detection. As future work, we plan to compare the vectors aligned with multilingual word embeddings after generation using MUSE with FastText pre aligned word embeddings. We can also explore BERT embeddings and evaluate their performance on the same task.

References

David Bamman and Noah Smith. 2015. Contextualized sarcasm detection on twitter.

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

Özlem Çetinoğlu, Sarah Schulz, and Ngoc Thang Vu. 2016. Challenges of computational processing of code-switching. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

Brenda Danet and Susan Herring. 2007. *The Multilingual Internet: Language, Culture, and Communication Online*.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, page 107–116, USA. Association for Computational Linguistics.

Sakshi Gupta, Piyush Bansal, and Radhika Mamidi. 2016. Resource creation for hindi-english code mixed social media text.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Lichan Hong, Gregorio Convertino, and Ed Huai hsin Chi. 2011. Language matters in twitter: A large scale study. In *ICWSM*.

Aditya Joshi, Pushpak Bhattacharyya, and Mark J. Carman. 2017. Automatic sarcasm detection: A survey. *ACM Comput. Surv.*, 50(5).

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Amit Mandelbaum and Adi Shalev. 2016. Word embeddings and their use in sentence classification tasks.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Stella Mónica, Mónica Cárdenas-Claros, and Neny Isharyanti. 2009. Code switching and code mixing in internet chating: betwen "yes", "ya", and "si" a case study. *The jaltcall Journal*, Vol 5:67–78.

Adithya Rao and Nemanja Spasojevic. 2016. Actionable and political text classification using word embeddings and lstm. *ArXiv*, abs/1607.02501.

Sahil Swami, Ankush Khandelwal, Vinay Singh, Syed Sarfaraz Akhtar, and Manish Shrivastava. 2018. A corpus of english-hindi code-mixed tweets for sarcasm detection.

Joseph Tepperman, David R. Traum, and Shrikanth S. Narayanan. 2006. "yeah right": sarcasm recognition for spoken dialogue systems. In (Joshi et al., 2017).

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas. Association for Computational Linguistics.

T. Young, D. Hazarika, S. Poria, and E. Cambria. 2018. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75.

Zhou Yu, Vikram Ramanarayanan, David Suendermann-Oeft, Xinhao Wang, Klaus Zechner, Lei Chen, Jidong Tao, Aliaksei Ivanou, and Yao Qian. 2015. Using bidirectional lstm recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech. pages 338–345.