SENTIMENT ANALYSIS AND TRANSFER LEARNING

Nobel Jacob Varghese
Saarland University
66123 Saarbrücken
noja00001@uni-saarland.de

Nitish Juttu
Saarland University
66123 Saarbrücken
niju00001@uni-saarland.de

March 31, 2021

ABSTRACT

Sentiment analysis or opinion mining is the computational study of people's opinions, sentiments, attitudes, and emotions. It is used to determine whether a piece of text contains some subjective information and what subjective information it expresses, that is whether the attitude behind this text is positive, negative or neutral and understanding the opinions of user-generated content is of great help for commercial and political use etc.In this project, we create a neural sentiment classifier by training on one particular language and then through the concept of transfer learning we apply and test on related different data set and also finally look into different approach for sentiment classification and analysis.

1 Introduction

Sentiment analysis also known as opinion mining or emotion AI refers to the use of natural language processing, text analysis, computational linguistics, and bio-metrics to systematically identify, extract, quantify, and study effective states and subjective information. Transfer learning is a machine learning research problem that focusing on storing knowledge gained while solving one problem and applying it to a different but related problem.Sentiment analysis has three main levels: document level, sentence level and aspect level. In this paper we will focus on sentence level which is concerned with the sentiment of each sentence.There are numerous approaches and techniques that can be utilised to classify the sentiment of the text and these can be either supervised, unsupervised or hybrid methods and even neural network techniques can be applied.Deep learning is one of the most common and powerful machine learning techniques that has been widely applied to sentiment analysis and shows great potentials and impacts on the performance of sentiment analysis.Deep learning is a machine learning method that uses artificial neural networks in learning tasks using networks of many hidden layers. In NLP, Deep learning models need word embedding which is a type of word representation, the words are transformed into vectors. There are different models used for word embedding.One of commonly used word embedding models is Word2vec. Also, we explore other data representation technique such as term frequency and inverse document frequency and then apply classification algorithm such as Support Vector Machine and then the results were measured using precision, recall, accuracy and F measure for evaluating the effectiveness of the proposed method.The remainder of this paper is organized as follows; Section 2 describes related works performed by other researchers in this field; Section 3 describes the proposed method to perform the experiment; and Section 4 describes the primary results and discussion obtained from the experiments. The last section presents the conclusions and suggested future work.

2 Methodology

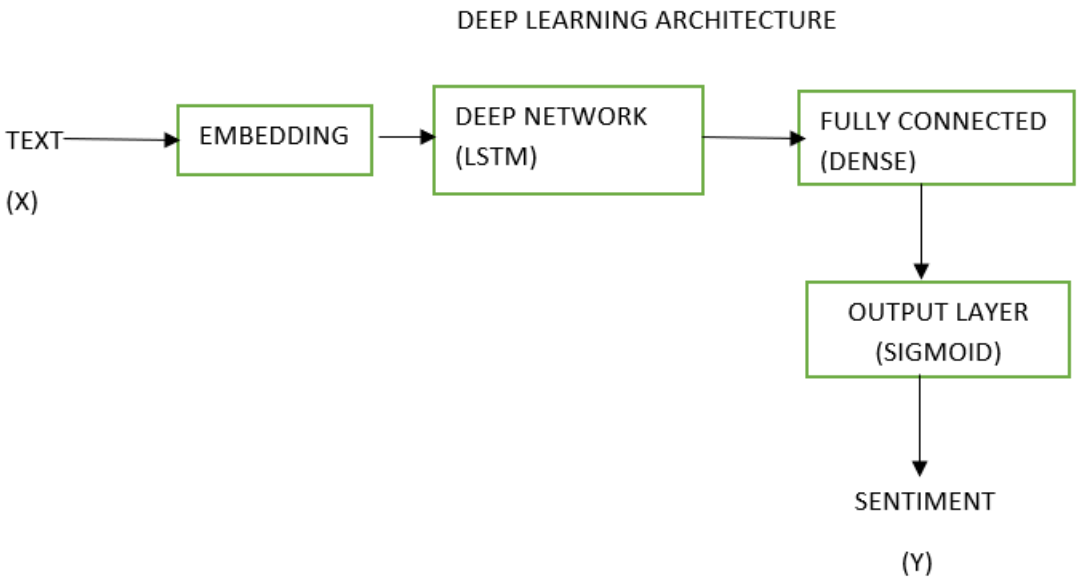
Neural networks work on numeric data and not on string or characters.To train neural networks and any machine learning model on text data, we need to convert the text data in some form of numerical representations before feeding it to the model or algorithm. Firstly we look into neural word embedding technique of word2vec using skip grams.A word embedding or word vector is a vector representation of an input word that captures the meaning of that word in a semantic vector space.In this skip gram model, we will use Hindi sentiment data set and then train neural network to extract word embeddings for the data.

Task Modeling: We took the Hindi dataset for word embedding and then train word2vec model.Firstly we considered only 100 data points by considering first 100 rows as our data.Then we pre-processed this Hindi data set by removing punctuations, lowering the text and removing the stop words from the data set and then compiled a list of all words from the corpus which will be utilised for building the vocabulary.The input to the first layer of word2vec is an one-hot encoding of the current word, so we generate one-hot encoding of an arbitrary word in the vocabulary.We also perform subsampling by taking the probability to keep a word using formula as given below.

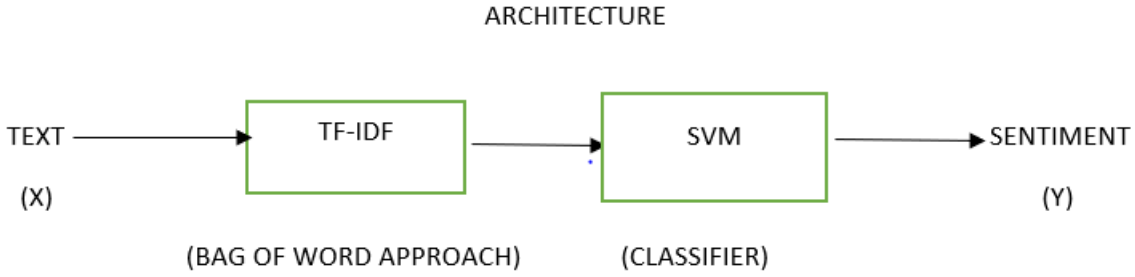
$$P_{keep}(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

where $z(w_i)$ is the relative frequency of the word w_i in the corpus and then we calculate word frequencies and then based on estimated probability we will retain the word in a given context.Now we will generate the Skip gram model that requires training data which considers the current word and context word.The context word refers to the words before and after current words given within a window size.In our model, we define window size of 5 and embedding size of 300 with reference to [1].The output of the model is then compared to a numeric class label of the words within the size of the skip-gram window.We define following hyperparameter for training.The learning rate alpha is chosen to be 0.025 because we followed the strategy that small learning rate may allow the model to learn a more optimal set of wieghts but may take longer time for training rather a large learning rate that will allow model to learn faster but at the cost of arriving on optimal solution.The number of epochs that we chose is 100 and the optimizer that we use for training model is stochastic gradient descent(SGD) taken with reference to [2]. The loss function that we deploy for training is negative log-likelihood (NLL) because we achieved convergence almost to zero as this worked well for our training and factor why we used negative log likelihood because as the output layer of neural network in skip gram is softmax function, the NLL loss function works well and it is mostly used in tandem with softmax function[3]. Finally we train our model and generate the Hindi embeddings.

Next we analyse deep learning based approach and develop neural sentiment analysis and perform transfer learning to check how model works on another natural language.We will utilise the generated Hindi word embeddings from above task for implementing binary neural sentiment classifier for Hindi section of the corpus.Here, firstly we preprocess the Hindi data, calculate the number of unique words for the vocabulary size and also we load the Hindi word embeddings.The next step is to divide dataset for training and validation.We took 80 percent as train data and 20 percent for testing.Then tokenizer is created for entire vocabulary and fit these tokenizer for each train and test dataset.The next step that we followed was to find the length of longest sentence and add padding based on the longest sentence.Then we define the embedding matrix and also define a function that returns corresponding index of the the word from task1 and the idea is to map the words in task2 to their corresponding weights in task1.Once we have everything in the pipeline, we then build a deep model or neural network to solve the problem.We have utilised Long Short-Term Memory layer(LSTM) model for our training and dropout rate of 0.1(set 10 percent of inputs to zero) and retain 90 percent.Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network. This has the effect of reducing overfitting and improving model performance, this wee have implemented using keras module available in python.Activation function followed in our model is sigmoid function because it exist between 0 and 1, it is especially used for models where we have to predict the probability as an output.Binary cross entropy loss function is utilised as we have binary classification task problem as these tasks need to answer a question with only two choices(0 or 1), also as we have used sigmoid activation function for traing the model, it is the only activation compatible with the binary crossentropy loss function.Adam optimizer is used[4] as it is a replacement optimization algorithm for stochastic gradient descent for training deep learning models.Also, adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm, also we had used SGD for our task1, considering all these factors we approached with adaptive optimizers for training our model.Then we evaluate on the test set and report accuracy score metrics. Overall, LSTM is used because with Long Short-Term Memory neural networks, we are introducing more more controlling knobs, which control the flow and mixing of inputs as per trained weights, giving us the most control-ability and thus,better results.



For the second part of task 2, firstly we look for total rows of Hindi data frame and as there are totally 4665 rows of Hindi dataset, we split Bengali corpus accordingly and take around 4500 rows of Bengali data such that it roughly equals the Hindi corpus in size and distribution of classes (hatespeech/non-hatespeech).The similar preprocessing strategy that we followed for above task 1 is followed here except without removal of Bengali stopwords and removal of emoji or emoticons as we only tried to consider textual part rather considering emoticons and ignored it.After preprocessing the data, we generate word embeddings for Bengali dataset, following similar procedure that was followed for task 1 with same window size and similar hyper parameters etc.Lastly in task2 we develop neural sentiment classifier using Bengali word embeddings in similar fashion as mentioned above and retrain it with Bengali data and report the accuracy scores.We ran in google colab and saved the corresponding weights using torch module but for displaying results and implementation in notebook, we have taken totally 200 rows in the jupyter note and then ran it.



Another different approach that we followed for sentiment analysis and classification is based on [5], that is using support vector machine.We tried to implement the paper [5] which represents different model and approach to solve sentiment analysis.We have used different model architecture with different data representation technique for our text data as we have used one of the feature extraction technique from the paper [5] which is TF-IDF scheme for text to numeric feature generation and then deploy SVM for predicting the user sentiment.Here we have used Bag of Words approach with TF-IDF(Term Frequency- Inverse Document Frequency) scheme, in order to convert text to numbers.Conceptually in the bag of words approach the vocabulary of all the unique words in all the documents is formed and this vocabulary serves as a feature vector.In a simple bag of words, every word is given equal importance but the idea behind TF-IDF is that the words that occur more frequently in one document and less frequently in other documents should be given more importance as they are more useful for classification.TF-IDF is a product of two terms which includes TF and IDF.Term Frequency is equal to the number of times a word occurs in a specific document. It is calculated as:

TF = (Frequency of a word in the document)/(Total words in the document)

Inverse Document Frequency for a specific word is equal to the total number of documents, divided by the number of documents that contain that specific word. The log of the whole term is calculated to reduce the impact of the division and it is calculated as:

IDF = Log((Total number of docs)/(Number of docs containing the word))

We have utilised the TfidfVectorizer class from the sklearn.feature extraction.text module to create feature vectors containing TF-IDF values.The attribute max features specifies the number of most occurring words for which you want to create feature vectors. Less frequently occurring words do not play a major role in classification. Therefore we retain 3000 most frequently occurring words in the dataset. The max df value of 0.7 percent specifies that the word must not occur in more than 70 percent of the documents.The rationale behind choosing 70 percent as the threshold is that words occurring in more than 70 percent of the documents are too common and are less likely to play any role in the classification of sentiment.Finally, to convert your dataset into corresponding TF-IDF feature vectors,we need to call the fit transform method on TfidfVectorizer class and pass it to our prepossessed dataset.We also divide our dataset into training and testing set before building the actual sentiment analysis model.We took 80 percentage of the data as train set and 20 percent as our test set.After dividing our data into training and test sets,next step that followed was to train the model on the training set and evaluate its performance on the test set.We have used SVM classifier to train our model.Linear kernal is utilsed because in our dataset as we have only only two classes(hate/non-hate) with positive and negative sentiment(labeled as 0 and 1) and it is useful when there is large number of features as compared to the training samples which is a suitable choice for our dataset and also linear SVM is less prone to overfitting than non-linear[6].For training the model,we call "fit" method on the classifier object and pass it the training feature set and training label set and then to make predictions on the test set, we pass the test set to the "predict" method. Finally for evaluation of our classification model, we use confusion matrix, classification report and accuracy as our performance metrics and these metrics can be calculated using classes from sklearn.metrics module.

3 Results

The accuracy value represents the percentage of test data which were classified correctly by the model.First screenshot shows loss function values for task 1 (hindi word embeddings) implementation.

```
loss is tensor(-1.2726e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.2281e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.1864e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.1475e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.1110e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.0767e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.0445e-07, grad_fn=<NllLossBackward>)
loss is tensor(-1.0141e-07, grad_fn=<NllLossBackward>)
loss is tensor(-9.8552e-08, grad_fn=<NllLossBackward>)
loss is tensor(-9.5853e-08, grad_fn=<NllLossBackward>)
loss is tensor(-9.3301e-08, grad_fn=<NllLossBackward>)
loss is tensor(-9.0885e-08, grad_fn=<NllLossBackward>)
loss is tensor(-8.8596e-08, grad_fn=<NllLossBackward>)
loss is tensor(-8.6423e-08, grad_fn=<NllLossBackward>)
loss is tensor(-8.4357e-08, grad_fn=<NllLossBackward>)
loss is tensor(-8.2392e-08, grad_fn=<NllLossBackward>)
loss is tensor(-8.0519e-08, grad_fn=<NllLossBackward>)
loss is tensor(-7.8730e-08, grad_fn=<NllLossBackward>)
loss is tensor(-7.7022e-08, grad_fn=<NllLossBackward>)
loss is tensor(-7.5386e-08, grad_fn=<NllLossBackward>)
loss is tensor(-7.3820e-08, grad_fn=<NllLossBackward>)
Training finished
```

Second screenshot shows accuracy values for task 2a implementation of Binary neural sentiment classifier with Hindi embeddings and Hindi dataset.Here we achieve 78 percent of accuracy when run for 10 epochs and validated on test data.

```
Epoch 1/10
3/3 [==============================] - ETA: 2s - loss: 0.6932 - accuracy: 0.59 - 2s 201ms/step - loss: 0.6926 - accuracy: 0.619
5 - val_loss: 0.6864 - val_accuracy: 0.8000
Epoch 2/10
3/3 [==============================] - ETA: 0s - loss: 0.6870 - accuracy: 0.78 - 0s 33ms/step - loss: 0.6856 - accuracy: 0.7859
- val_loss: 0.6789 - val_accuracy: 0.8000
Epoch 3/10
3/3 [==============================] - ETA: 0s - loss: 0.6792 - accuracy: 0.78 - 0s 31ms/step - loss: 0.6784 - accuracy: 0.7742
- val_loss: 0.6699 - val_accuracy: 0.8000
Epoch 4/10
3/3 [==============================] - ETA: 0s - loss: 0.6704 - accuracy: 0.78 - 0s 28ms/step - loss: 0.6695 - accuracy: 0.7742
- val_loss: 0.6584 - val_accuracy: 0.8000
Epoch 5/10
3/3 [==============================] - ETA: 0s - loss: 0.6634 - accuracy: 0.75 - 0s 30ms/step - loss: 0.6592 - accuracy: 0.7625
- val_loss: 0.6430 - val_accuracy: 0.8000
Epoch 6/10
3/3 [==============================] - ETA: 0s - loss: 0.6670 - accuracy: 0.65 - 0s 84ms/step - loss: 0.6474 - accuracy: 0.7430
- val_loss: 0.6222 - val_accuracy: 0.8000
Epoch 7/10
3/3 [==============================] - ETA: 0s - loss: 0.6106 - accuracy: 0.84 - 0s 33ms/step - loss: 0.6173 - accuracy: 0.7977
- val_loss: 0.5911 - val_accuracy: 0.8000
Epoch 8/10
3/3 [==============================] - ETA: 0s - loss: 0.5686 - accuracy: 0.84 - 0s 34ms/step - loss: 0.5858 - accuracy: 0.7898
- val_loss: 0.5452 - val_accuracy: 0.8000
Epoch 9/10
3/3 [==============================] - ETA: 0s - loss: 0.5977 - accuracy: 0.71 - 0s 31ms/step - loss: 0.5556 - accuracy: 0.7586
- val_loss: 0.4965 - val_accuracy: 0.8000
Epoch 10/10
3/3 [==============================] - ETA: 0s - loss: 0.4776 - accuracy: 0.81 - 0s 32ms/step - loss: 0.5094 - accuracy: 0.7898
- val_loss: 0.5069 - val_accuracy: 0.8000
<tensorflow.python.keras.callbacks.History object at 0x000002E6663635C8>
```

Third screenshot shows accuaracy for task 2d implementation of binary neural sentiment classifier with Bengali embeddings and Bengali dataset.Here we obtain 40 percent of accuracy when tested with test data.We achieve low accuracy score maybe due to non removal or emoji or emoticons which conveys user's sentiment and emotions and these are not captured while doing the feature extraction, also usage of smaller bengali dataset maybe the cause for low accuracy score.One another important aspect what we noticed is that we slightly achieved better results while training with more number of epochs and increase in number of neural networks.

```
Epoch 1/10
5/5 [==============================] - ETA: 0s - loss: 0.6628 - accuracy: 0.62 - ETA: 0s - loss: 0.6840 - accuracy: 0.53 - 0
s 39ms/step - loss: 0.6840 - accuracy: 0.5312 - val_loss: 0.7042 - val_accuracy: 0.4000
Epoch 2/10
5/5 [==============================] - ETA: 0s - loss: 0.6942 - accuracy: 0.50 - ETA: 0s - loss: 0.6838 - accuracy: 0.53 - 0
s 30ms/step - loss: 0.6838 - accuracy: 0.5312 - val_loss: 0.7046 - val_accuracy: 0.4000
Epoch 3/10
5/5 [==============================] - ETA: 0s - loss: 0.7032 - accuracy: 0.40 - ETA: 0s - loss: 0.6838 - accuracy: 0.53 - 0
s 29ms/step - loss: 0.6838 - accuracy: 0.5312 - val_loss: 0.7049 - val_accuracy: 0.4000
Epoch 4/10
5/5 [==============================] - ETA: 0s - loss: 0.6913 - accuracy: 0.53 - ETA: 0s - loss: 0.6838 - accuracy: 0.53 - 0
s 30ms/step - loss: 0.6838 - accuracy: 0.5312 - val_loss: 0.7053 - val_accuracy: 0.4000
Epoch 5/10
5/5 [==============================] - ETA: 0s - loss: 0.7122 - accuracy: 0.31 - ETA: 0s - loss: 0.6836 - accuracy: 0.53 - 0
s 29ms/step - loss: 0.6836 - accuracy: 0.5312 - val_loss: 0.7056 - val_accuracy: 0.4000
Epoch 6/10
5/5 [==============================] - ETA: 0s - loss: 0.6995 - accuracy: 0.43 - ETA: 0s - loss: 0.6836 - accuracy: 0.53 - 0
s 29ms/step - loss: 0.6836 - accuracy: 0.5312 - val_loss: 0.7061 - val_accuracy: 0.4000
Epoch 7/10
5/5 [==============================] - ETA: 0s - loss: 0.6671 - accuracy: 0.59 - ETA: 0s - loss: 0.6834 - accuracy: 0.53 - 0
s 30ms/step - loss: 0.6834 - accuracy: 0.5312 - val_loss: 0.7074 - val_accuracy: 0.4000
Epoch 8/10
5/5 [==============================] - ETA: 0s - loss: 0.6821 - accuracy: 0.62 - ETA: 0s - loss: 0.6828 - accuracy: 0.53 - 0
s 29ms/step - loss: 0.6828 - accuracy: 0.5375 - val_loss: 0.7118 - val_accuracy: 0.4000
Epoch 9/10
5/5 [==============================] - ETA: 0s - loss: 0.6790 - accuracy: 0.65 - ETA: 0s - loss: 0.6754 - accuracy: 0.54 - 0
s 32ms/step - loss: 0.6754 - accuracy: 0.5437 - val_loss: 0.7493 - val_accuracy: 0.4000
Epoch 10/10
5/5 [==============================] - ETA: 0s - loss: 0.6592 - accuracy: 0.46 - ETA: 0s - loss: 0.6857 - accuracy: 0.54 - 0
s 31ms/step - loss: 0.6857 - accuracy: 0.5437 - val_loss: 0.7601 - val_accuracy: 0.4000
```

Forth screenshots represents accuracy value and others metrics for task 3.Here we get accuracy score of 87.26 percent for Bengali data and 74.91 percent for Hindi data.Also we have considered small set of data for our training and classification.We tried to implement based on the paper[5] and wanted to implement totally a different approach and see how it works for smaller set of data and we noticed here that we get better accuracy score value here, given small dataset with SVM classifier.

Screenshot for Bengali data:

```
Confusion Matrix:
 [[3802  219]
 [ 545 1434]]
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      4021
           1       0.87      0.72      0.79      1979

    accuracy                           0.87      6000
   macro avg       0.87      0.84      0.85      6000
weighted avg       0.87      0.87      0.87      6000

Accuracy Score:
 0.8726666666666667
```

Screenshot for Hindi data:

```
Confusion Matrix:
 [[336 101]
 [133 363]]
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.77      0.74       437
           1       0.78      0.73      0.76       496

    accuracy                           0.75       933
   macro avg       0.75      0.75      0.75       933
weighted avg       0.75      0.75      0.75       933

Accuracy Score:
 0.7491961414790996
```

4 Conclusion

Word2vec is one of the common models that is used in a word embedding. It is a prediction based algorithm that is used to represent a word as vectors with semantic relation. It is applied in many NLP tasks and has shown a great potential impact on the performance of sentiment analysis when used with deep networks.Another technique that was followed was TF-IDF based bag of words approach wherein we applied machine learning algorithm namely SVM as our classifier.We tried to implement holistic approach covering both word2vec skip gram technique with deep network and as well bag of words(BOW) based TF-IDF approach with machine learning algorithm based classifier (SVM) for sentiment analysis based on the paper[5].The key point that we learned is that skip gram based word2vec captures more context of the words and deep network works well for very large data set and TF-IDF based bag of word approach along SVM classifier works well for smaller data set with less consumption of resources and other parameters as compared to deep network training.

References

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26:3111–3119, 2013.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[3] http://cs231n.stanford.edu/

[4] https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456

[5] Nurulhuda Zainuddin, Ali Selamat. Sentiment Analysis Using Support Vector Machine. IEEE 2014 International Conference on Computer, Communication, and Control Technology (I4CT 2014)

[6] Henry Han, Xiaoqian Jiang. Overcome Support Vector Machine Diagnosis Overfitting Supplementary Issue: Computational Advances in Cancer Informatics (A)