Automatic sarcasm detection over social media

Name:	**Alli Khadga Jyoth**
Registration No./Roll No.:	19024
Institute/University Name:	IISER Bhopal
Program/Stream:	BS/DSE
Problem Release date:	August 08, 2022
Date of Submission:	November 30, 2022

1 Introduction

Sarcasm, a cutting or painful expression of irony, is frequently used to convey strong emotions like contempt, derision, or bitterness. Sarcasm is a way for people to make fun of or express contempt in a sentence or while speaking. Positive words are sometimes used to mask depressing emotions. Sarcasm and irony are often used in social media these days. Finding sarcasm is crucial to figuring out what individuals really think and feel. Many NLP applications, such as marketing research and opinion mining, can benefit from the use of sarcasm detection. However, sarcasm detection is an extremely challenging endeavor because it heavily depends on context, prior information, and the tone of the speech. The data set used for this project is the Self-Annotated Reddit Corpus (SARC). The corpus is a collection of Reddit posts. The train data set has 808661 rows and 4 columns (including the target class). The test set has 202166 rows and 3 columns. There are 2 class labels namely Sarcastic, and Non-Sarcastic in the train set. Each row of the training corpus contains the author id, comment, parent comment, and class label, whereas the test set just contains ids, comments, and parent comments of individual users. The dataset is a balanced dataset.

In this project, we have used Python libraries such as Numpy, Pandas, NLTK, TensorFlow, Keras, and Spacy.

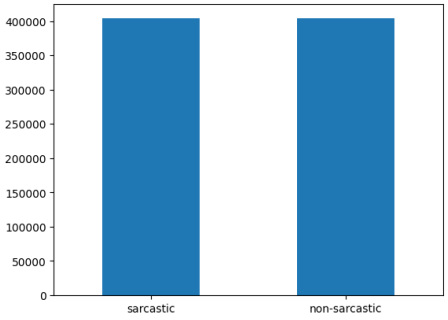

Figure 1: train dataset count

2 Methods

The below subsections represent the step-by-step work in order to complete this project.

2.1 Pre Processing

Before applying the Deep Learning models and feature extraction we did some pre-processing work discussed below :

- Lowering the texts so that “Go” and “go” represent the same features.
- Noise removal: We have removed numbers, newlines, and a single-word comment to speed up the feature mining process.
- In this corpus there are some rows whose values are meaningless or have unnecessary repeated characters like (“HAHAHAHAHAHAHAAHAHAHAHA”) which unnecessarily take more computational power and time. So to remove these words, we are taking the length of each word to

be 20 characters and any word having a length greater than 20 is replaced with a neutral word that has no meaning "OOV".

- We are also removing those words which have more numeric characters than the alphabet (e.g. "a12", "2a2", etc.) since these words don't have any meaning. Whereas keeping those words (e.g. "b4u", "2complicated4me", etc.,) that are commonly used in messages on social media nowadays and have meaning.
- Removing URLs, square Brackets, new line characters, @ and # symbols, tags, and punctuations.
- After this we are replacing emojis and emoticons with their respective meanings since emojis and emoticons play an important role in sentiment analysis and/or sarcasm detection.
- Nowadays people generally use words in contracted form (such as "I've" in place of I have, "luv" for love, "ne'er" for never, etc.). So we are expanding these contracted words into their full forms. Since in contracted form these words don't have any meaning for machine learning models.
- Performed Word Tokenization (i.e splitting a string or text into a list of tokens (words)) and removed all the commonly used words from the corpus that are insignificant for sarcasm detection (such as "the", "a", "an", "in", etc.) called Stopwords.
- Then we have done spelling corrections of incorrect words present in the corpus.
- After this we did Lemmatization (i.e grouping together the different inflected forms of a word so they can be analyzed as a single item, e.g, better $\rightarrow$ good and best $\rightarrow$ good) with POS (Part-of-Speech) tagging (i.e label assigned to each word to indicate the part of speech, tense, plural/singular, case, etc. e.g. driving is tagged with 'v' (i.e Verb) based on the context.)
- We filled those texts with "OOV" which only consists of meaningless and stopwords because after pre-processing those texts become "empty" or say have "NaN" values and on "NaN" values, we can't perform any NLP/ML techniques and models.
- After performing all these above steps we get a cleaned dataset. Then by using Keras we tokenize each sentence in the training set and select the top 5000 most frequent words to create a vocabulary. Now by using this vocabulary we convert each word from the first 200 words of each sentence into an integer. So that we can pass it to the model. Similarly, by this vocabulary, we convert test set also to integers.

2.2 Word embeddings

It is a technique where individual words are transformed into a vector. Where each word is mapped to one vector, this vector is then learned in a way that resembles a neural network. The vectors try to capture various characteristics of that word with regard to the overall text. In this project the words are embedded into a 200 dimensional vector for all the models used.

2.3 Deep Learning Models Used

In this project we have used some Deep Learning models namely: The Feed Forward Neural Network (FFNN), Stacked Recurrent Neural Network (RNN), Stacked LSTM, and Stacked Transformer, having 3-4 layers in each model, by using the Keras library. To apply these models to the dataset, we have divided the training dataset into train and validation sets in the ratio of 80/20 with the help of train_test_split from scikit-learn.

- **FFNN**: Feed-forward neural networks are artificial neural networks in which nodes do not form loops. Due to the fact that all input is simply sent forward, this kind of neural network is also referred to as a multi-layer neural network. Data flow involves the receipt of data at input nodes, transmission across hidden layers, and exit at output nodes. There are no linkages in the network that could be manipulated to convey data back from the output node. Shown in Fig.2

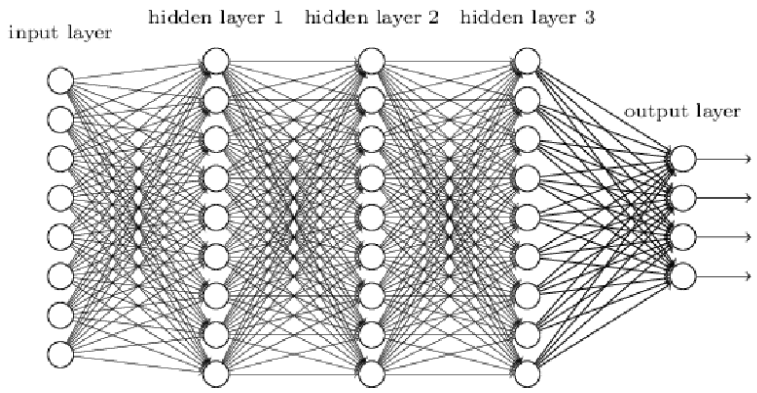

Figure 2: FFNN

- **RNN**: It is a type of Neural Network where the output from the previous step is fed as input to the current step i.e., connections between nodes have a cycle. Shown in Fig.3

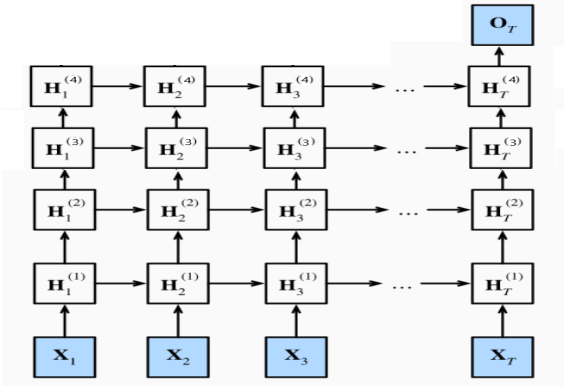

Figure 3: Stacked RNN

- **LSTM**: It is a type of RNN capable of learning order dependence in sequence prediction problems. Shown in Fig.4

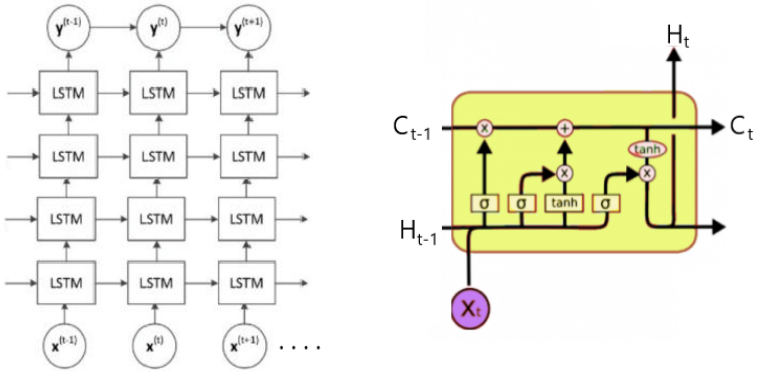

Figure 4: Stacked LSTM

- **Transformer**: It is a deep learning model that uses the self-attention process and weights the importance of each component of the input data differently. Shown in Fig.5

3 Evaluation Criteria

Accuracy is defined as the fraction of predictions our model predicts right.

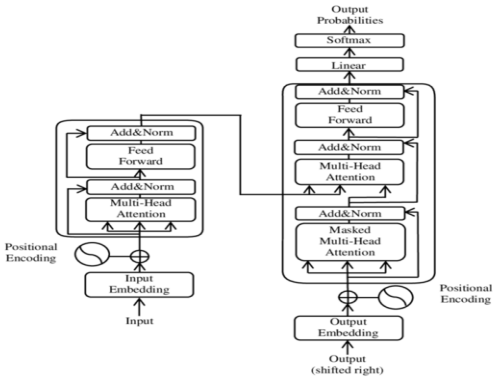

Figure 5: Transformer

$P = No.\ of correct\ predictions/Total\ No.\ of\ predictions$

Precision is defined as the number of true positives (Tp) over the number of true positives plus the number of false positives (Fp).

$P = Tp/(Tp + Fp)$.

F1 score is defined as the harmonic mean of precision and recall. $f1 = 2 * P * R/(P + R)$

4 Experimental Analysis:

In this section, we have listed the Accuracy, Precision, and f1-score measure of all the models.

Model	Accuracy	Precision	f1-score	parameters(≈)
FFNN	0.6271	0.6271	0.6272	3.6M
Stacked RNN	0.6521	0.6521	0.6521	1.1M
Stacked LSTM	0.6543	0.6543	0.6543	1.36M
Stacked Transformer	**0.6646**	**0.6646**	**0.6646**	3.02M

Table 1: Models Performance

5 Analysis of Results

From Table 1 we selected the best model as a Transformer based on accuracy. And then run it to predict the class labels of the test dataset. From the 1 we can see the effectiveness of recurrent models. Recurrent models like RNN and LSTM, although having only one-third the number of parameters compared to a FeedForward model, still achieved $\approx 2.3\%$better accuracy and f1-score. This can be directly attributed to their architecture differences, where the recurrent networks can store and process information from the past better than the FeedForward networks.

6 Discussions and Conclusion

The performance of the models may further be improved by using more range of hyperparameters and using other classification models which require more computational power.

7 Contribution

Akash Kumar Singh wrote the code for Pre-Processing, FFNN and Transformer whereas Alli Khadga Jyoth wrote the code for Stacked RNN and Stacked LSTM.

8 Refrences

- Fig.2 is taken from this link[1]
- Fig.5 is taken from this link[2]

[1] https://towardsdatascience.com/feed-forward-neural-networks-c503faa46620
[2] https://arxiv.org/abs/1706.03762