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:	September 29, 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 "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, Scikit-Learn, Matplotlib, NLTK, and Spacy.

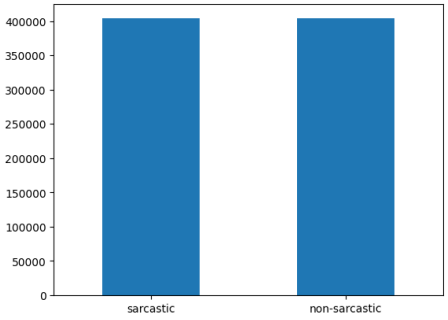

Figure 1: train dataset count

2 Methods

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

2.1 Pre Processing

Before applying Classification model and feature extraction we did pre-processing discussed below :

1. Lowering the texts so that "Go" and "go" represent the same features.
2. Noise removal: We have removed numbers, newlines, and a single-characters to speed up the feature mining process.
3. 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 capping the word length to 20.

4. We are also removing those words which have more numeric characters than the alphabets (e.g. "a12", "2a2", ...) since these words don't have any meaning. Whereas keeping those words (e.g. "b4u", "2complicated4me", etc.,) that are commonly used on social media nowadays and have meaning.

5. Removing URLs, square Brackets, new line characters, @ and # symbols, tags, punctuations.

6. 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.

7. 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, they all represent the same word.

8. Performed Word Tokenization (i.e splitting a string, 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.

9. Then we have done spelling corrections of misspelled words present in the corpus.

10. 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 → good) with POS (Part-of-Speech) tagging.

11. 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.

2.2 Text Feature Extraction

One of the biggest problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured well-defined fixed-length inputs. ML algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers. So we have used the following Feature Extraction Techniques to convert variable-length texts into a fixed-length vector:

Bag of Words: A bag of words is a representation of text that describes the occurrence of words within a document. It creates Document Vectors of fixed length for each document.

Tf-idf: It is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

$$w_{i,j} = tf_{i,j} \times log\left(\frac{N}{df_i}\right)$$

where, $tf_{i,j}$ = number of occurrences of word i in document j
df_i = number of documents containing word i
N = total number of documents

Word embeddings: It is a technique where individual words are transformed into a vector. The vectors try to capture various characteristics of that word with regard to the overall text. In this project we have used Pre-trained Word embeddings by SPACY and our own trained word embeddings.

2.3 Classification Models Used & Hyper-Parameter Tuning

In this project we have used some classification models namely : Logistic Regression, Decision Tree, Random Forest, Multinomial Naive Bayes, Support Vector Machine (Linear), with hyperparameter tuning using Scikit-Learn library.

2.4 Feature Scaling

We have applied MinMaxScaler, to sclae the input vector to [0,1] because Multinomial Naive Bayes takes only positive inputs and embedding models returns vectors with negative values.

2.5 Feature Selection

Having irrelevant features in the dataset decrease the performance of the models. To overcome this we applied χ^2 Feature Selection technique on the dataset, to get the k best features.

3 Evaluation Criteria

Precision is defined as the number of true positives (Tp) over the number of true positives plus the number of false positives (Fp). $P = Tp/(Tp + Fp)$.

Recall is defined as the number of true positives (Tp) over the number of true positives plus the number of false negatives (Fn). $R = Tp/(Tp + Fn)$.

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

Micro averaged is defined as sum of true positives for all the classes divided by the sum of all true positives and false positives. We evaluate the performance of our model based on F1-macro score.

4 Analysis of Results

Each of the 5 models' performance in Bag-of-words, TfIDF, Spacy pre-trained word embedding and Word2Vec custom trained embedding was measured and recorded after feature selection and hyperparameter tuning using GridSearchCV. Out of all the models, Logistic Regression achieved the highest performance with an accuracy 64.81 and a F1 score of 64.47 in Bag-of-Words. The highest performance was achieved with k=10000 features with χ^2 statistics. And then run the best set of combinations to predict the class labels of test dataset.

Models	Accuracy	F1-Score
Logistic Regression (SGD)	**64.81**	**64.47**
Decision Tree	60.44	60.08
Random Forest	62.35	62.06
Linear SVC (SGD)	64.16	63.33
Multinomial NB	64.56	64.04

Table 1: Bag of Words

Models	Accuracy	F1-Score
Logistic Regression (SGD)	62.64	62.45
Decision Tree	61.16	60.2
Random Forest	63.32	62.84
Linear SVC (SGD)	63.65	63.51
Multinomial NB	64.21	64.15

Table 2: Tf-IDF

5 Discussions and Conclusion

The Logistic Regression achieved the highest performance in Bag-of-Words, but it is noticeable that it was also the top performing model in other methods also. This shows the versatility of linear classifier Logistic regression, which is implemented using SGDClassifier in our approach. This divergence in our approach to Logistic Regression is because, the inability of Logistic Regression model in sklearn

Models	Accuracy	F1-Score
Logistic Regression (SGD)	60.1	60.04
Decision Tree	56.28	56.28
Random Forest	58.63	58.24
Linear SVC (SGD)	59.63	59.54
Multinomial NB	56.01	56.01

Table 3: Spacy Pretrained

Models	Accuracy	F1-Score
Logistic Regression (SGD)	55	54.3
Decision Tree	55.05	54.48
Random Forest	56.43	56.36
Linear SVC (SGD)	52.42	45.25
Multinomial NB	52.6	52.5

Table 4: Word2Vec trained

to directly work with sparse matrices. This constraint is alleviated by using the SGDClassifier with its loss set to "log_loss". The low performance of our trained embedding model can be explained by the low amount of training data available to train the embedding. Our data only contains only 1 million sentences, which is lot of data when compared to other types of classification settings, but in the context of NLP its not enough. The pretrained Spacy model also didn't perform well because, the embedding is trained on a general corpus, but the task at hand requires a deep understanding of the word representations within the corpus. So, the pretrained embedding models fail to capture the corpus dependent word representations and therefore performs worse than the regular Bag-of-words and Tf-IDF methods. The performance of the models may further be improved by using more range of hyper parameters and using other classification models which require more computational power.

6 Contributions

I have implemented the custom trained Word2Vec embedding. It was trained on the whole corpus of training data. The embedding gives a 300 dimension vector for each uni-gram present in the corpus. The similarity between two vectors represent the similarity between two word representations. I have also implemented the Tf-IDF methods. For preprocessing I took the job of manual comments checking and dealing with outlier comments like repeated texts and excessive length comments. I have done some part of writing code for classification pipelines.