

# SARCASM DETECTION IN NATURAL LANGUAGE PROCESSING

## INTRODUCTION

Sarcasm is a prevalent form of communication in human interactions, often used to convey irony, humor, or criticism. Recognizing and understanding sarcasm in textual data is a challenging task for natural language processing (NLP) systems. In recent years, the advent of deep learning has significantly advanced the field of NLP, opening new possibilities for detecting and interpreting sarcasm in text. Sarcasm detection can benefit natural language generation tasks. By recognizing sarcastic intent, NLP models can generate responses that align with the intended tone, making human-computer interactions more natural and effective.

In summary, learning and improving sarcasm detection in NLP have significant practical applications, ranging from sentiment analysis and social media analysis to enhancing language understanding and natural language generation capabilities. Advancements in sarcasm detection contribute to more accurate and nuanced language processing, leading to improved performance across various NLP tasks and applications.

## Applications and Use Cases

Sarcasm detection plays a crucial role in comprehending individuals' genuine sentiments and opinions. The application of sarcasm detection techniques holds significant advantages across various domains within NLP, such as marketing research, opinion mining, and information categorization. By accurately identifying sarcasm in text, valuable insights can be gained, enabling more accurate analysis of customer feedback, public sentiment, and the categorization of information for effective decision-making and understanding trends in different fields.

Indigo Airlines, faced significant negative attention due to a Twitter response to a customer's sarcastic tweet about misplaced baggage. The airline, known for its dominant market share in India's domestic air travel and its claim of superior on-time performance, unfortunately responded by thanking the disgruntled customer. This incident quickly went viral, leading to public backlash before Indigo eventually deleted the tweet. This highlights the need for sarcasm detection in customer feedbacks and reviews which are increasingly first responded to by an AI bot.



## Background

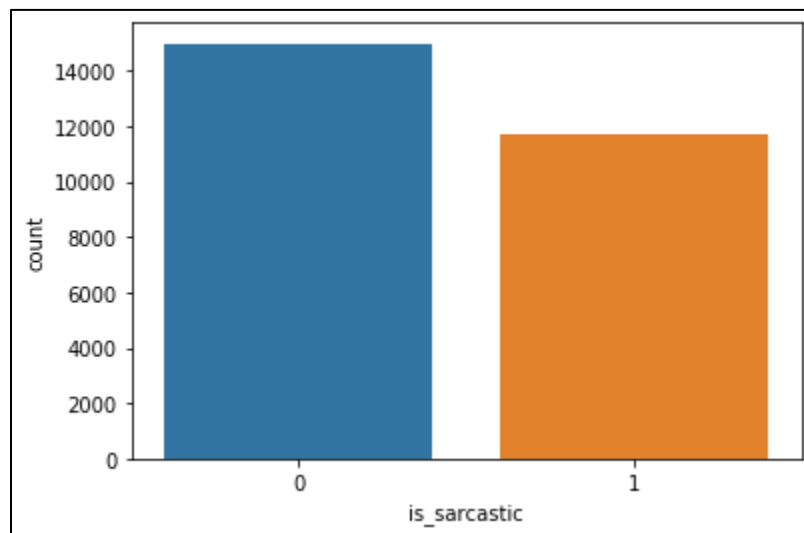
Our main objective is to determine if a sentence is sarcastic or non-sarcastic. Considering it is mainly based on the context of utterances or sentences, it is difficult to design a model to adroitly detect sarcasm in the natural language processing (NLP). Testing on the words will not suffice to detect sarcasm. In such cases, factors like changes in tone, overemphasis of words, and drawn-out syllables help to decide whether it is sarcastic or not. Lexical and pragmatic features also play a crucial role in the detection of sarcasm.

### NEWS HEADLINES DATASET:

The News Headlines dataset is collected from two news websites, sarcastic news headlines from theonion.com and non-sarcastic headlines from huffingtonpost.com. TheOnion specifically publishes sarcastic versions or humorous takes in their news headlines becoming the positive class for our two class dataset.

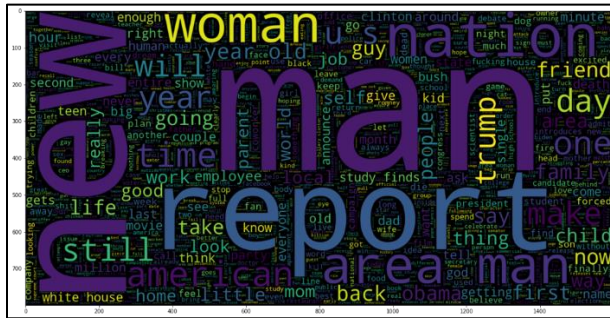
## Dataset

The News Headlines dataset is a relatively balanced dataset with 11724 sarcastic headlines out of a total 26709 headlines which comes to around 44%. It is unique in this sense because a lot of other datasets especially the ones extracted from twitter show a much lower positive class representation because of less frequency of sarcasm in the social media corpus. Having equal representation of sarcasm allows us to effectively learn the features and characteristics to be derived out of the two set of textual entities.

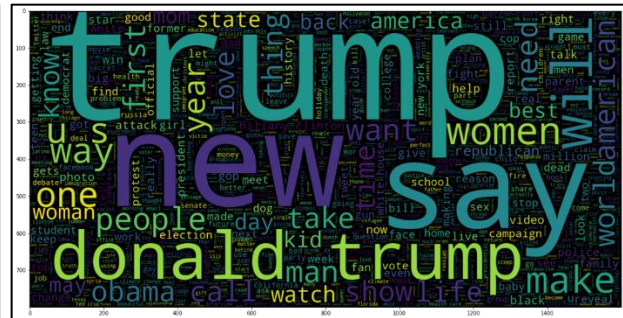


Looking at the wordcloud below, we observe no major differentiation in the words that come up frequently apart from a few named entities which occur more frequently in the actual news headlines naturally due to the nature of reporting and journalism. This suggests it is not the choice of words that conveys sarcasm, rather something much more complex.

Wordcloud for Non-sarcastic headlines



Wordcloud for sarcastic headlines:



#### Reason for choosing the News Headlines Dataset:

- The dataset is collected from two news websites, sarcastic news headlines from theonion.com and non-sarcastic headlines from huffingtonpost.com. TheOnion specifically publishes sarcastic news thus giving high quality labels for the positive class.
- As news headlines are written by professional journalists, they exhibit fewer spelling mistakes and junk words including the informal lingo in case of social media-based datasets such as Semeval or SARC. Consequently, the percentage of words for which we were able to locate word2vec embeddings within the News Headlines Dataset (77%), was higher than the percentage found in the Semeval Twitter-based dataset (64%).

Statistic/Dataset	Headlines	Semeval	IAC	SARC
# Records	28,619	4792	3260	1,010,826
Domain	News	Twitter	Debate	Reddit
Labeling technique	Source-based	Hand labeled	Hand labeled	Tag-based
Label quality	Controlled	Controlled	Controlled	Not controlled
Language	Formal	Informal	Formal	Informal
% word2vec embeddings found	77	64	–	–

h

#### State of the Art – Literature survey

Apart from the Twitter database there are various other datasets also used in much research like Internet Argument Corpus (IAC), SARC, SemEval 2018 task 3, and SemEval 2015 Task 11. Similarly, the news headline dataset used by Mandal comprises of 26,709 news headlines. Among these headlines, 43.9% were satire, and 56.1% were real. They claimed that no other works had used that dataset to train neural networks yet, also had explained CNN-LSTM-based architecture with detail and reached 86.16% accuracy. Du et al. [10] found Rhetoric irony in the corpus.

Cutting-edge research papers play a vital role in the progress of sarcasm detection within state-of-the-art facilities. A notable example is the study titled "Hierarchical Attention-Based RNN for Sarcasm Detection" authored by X. Zhang et al. in 2018. The paper introduces an innovative method that combines context-based models and sequential models, specifically utilizing the IITB sequential model. By integrating these approaches, the detection of sarcasm in textual data becomes more accurate. The context-based models utilize contextual information to capture the subtle nuances of sarcasm, while the sequential model, developed by the Indian Institute of Technology Bombay (IITB), effectively identifies the sequential patterns and dependencies within the text. This amalgamation of techniques enables the

creation of advanced facilities capable of robust sarcasm detection, ultimately contributing to the advancement of natural language comprehension and sentiment analysis.

### **Limitation of the state of the art.**

Sarcasm heavily relies on context, making it challenging for models to accurately capture and interpret the intended sarcastic meaning without a thorough understanding of the surrounding context. Models may struggle with context-dependent sarcasm or cultural nuances that affect the perception of sarcasm. Developing sarcasm detection models requires large-scale labeled datasets that accurately represent the diverse range of sarcastic expressions. However, acquiring and annotating such data can be labor-intensive and time-consuming, limiting the availability of high-quality labeled datasets for training sarcasm detection models. Sarcasm can also be expressed through the vocal tones and voice modulations, a person annotating textual representation of one's remarks may vary in the intensity of visible sarcasm and therefore, adding subjectivity to the annotated dataset.

Sarcasm detection can indeed become more challenging as the length of the text increases. Longer texts often contain more context and complex linguistic structures, making it harder to identify sarcastic elements accurately. Sarcasm can rely on subtle cues, such as tone, irony, or contrasting statements, which may be more difficult to detect within lengthy passages. Additionally, longer texts may involve a mix of sarcastic and non-sarcastic elements, requiring more comprehensive analysis and context understanding. However, with advanced natural language processing techniques and effective feature extraction methods, it is possible to overcome these challenges and improve sarcasm detection performance even in longer texts.

### **Application (in a domain of your choice)**

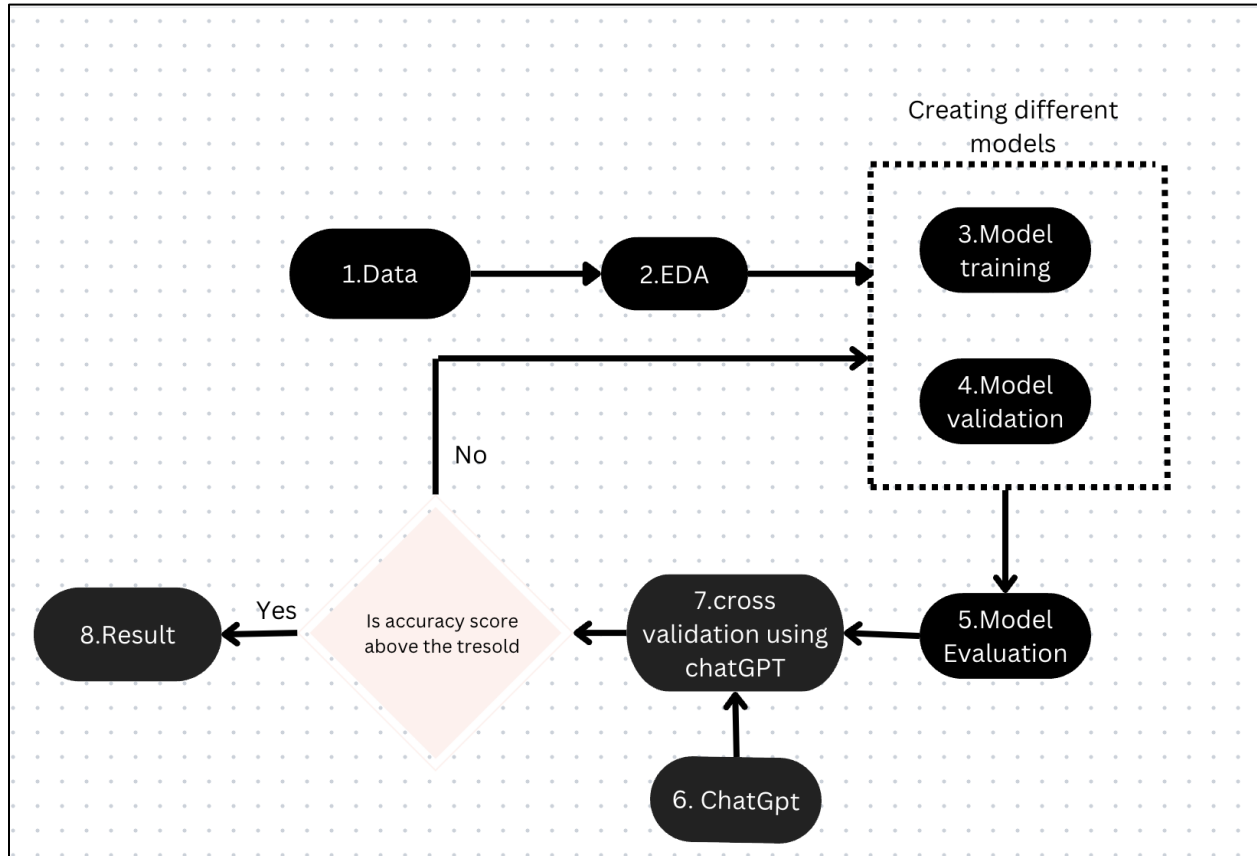
#### Reputation Management in BFSI: (Text Sarcasm Detection):

- Customer sentiment monitoring on social media feeds – Sarcasm detection can be used to monitor social media text with mentions of the bank's name, branch, or other metadata. This can help the BFSI institutes respond sooner and avoid any potential brand damage.
- News Monitoring – The BFSI organization can monitor the news to spot sarcastic mentions and act appropriately.
- Customer Feedback Monitoring - Sarcasm detection can be utilized in analyzing customer feedback surveys, reviews, or feedback forms. By identifying sarcastic comments, businesses can gain deeper insights into customer sentiments and experiences, enabling them to address concerns, improve products or services, and enhance overall customer satisfaction.
- Email sarcasm detection – Sarcasm can be a strong indicator of employee stress. If more sarcasm is detected in a team's communications etc. Timely interventions can be made. This of course needs to be done in a manner that does not infringe personal data.
- Embedded in BFSI chatbots – Can help detect incoming text of sarcastic nature and can respond appropriately.
- Sarcasm generation can also be used intelligently to run a specific market campaign to come up with witty taglines and advertisements.

- Online Banking alerts – Most people do not respond or attend to messages sent by BFSI organizations. Adding sarcasm to the text can get them to focus and therefore make it stick more.

By detecting sarcasm, businesses can gain insights into consumer opinions, preferences, and sentiments, helping them make informed business decisions.

## METHODOLOGY



The following steps explains the end-to-end process shown in above figure.

1. **Data:** We got two dataset one from the Kaggle sarcastic news headline dataset and another one is tweets datasets. We loaded data into Colab from Google Drive. The same directory also contains the pretrained Glove and Word2vec embeddings to be used later.
2. **EDA:** After loading these two datasets, we have done the EDA and found that the quality of the data in tweets dataset is very poor which causes overfitting issue and it only performed well on avery specific set of data hence we decided to use the sarcastic headline dataset for all the models. The dataset has around 26k rows.
3. **Model Training:**
  - Then dataset into two training and validation dataset as the ratio of 80% for training and 20% for validation and then model is trained using the training dataset and then model is validated using the validation dataset.

- There are 4 types of models are trained they are Logistic Regression, LSTM-RNN, Bidirectional LSTM with Glove and Word2Vec embeddings and Naïve bayes classifier.
4. **Hyperparameter Tuning:** For hyperparameter tuning, we used GridSearch module to find the optimal number of epochs, batch size, learning rate etc. Also used EarlyStopping conditions to stop training if the model accuracy on the validation data has not improved since last 3 epochs, and Checkpoint is saved with the weights when the validation accuracy is highest among the epochs. All the trained models are saved in the drive repository.
  5. **Model Evaluation:** All the 4 models are then evaluated, and the best model is chosen based on the evaluation metrics and their training and validation loss.
  6. **Cross validation using ChatGPT:** Sarcastic and non-Sarcastic data are then fetched from chatGPT to cross verify the model. We call the ChatGPT API using the openai python package and ask it to return a sarcastic or non-sarcastic news. We feed this response into our trained models and evaluate.

## MODELS AND ARCHITECHURE:

### LOGISTIC REGRESSION

We have defined Logistic Regression as our baseline model for this text classification task. We will be building complex models ahead to increase the performance of the baseline.

S.No.	Model Name	Testing Score
1.	Linear Support Vector Classifier	83.75
2.	Gaussian Naïve-Bayes	73.80
3.	Logistic Regression	83.08
4.	Random Forest Classifier	79.71

Baseline researches using the standard models on the News Headlines dataset are shown above. Our aim is to improve on these metrics using LSTM and embeddings.

### LSTM-RNN

LSTM, a recurrent neural network (RNN) architecture widely used in deep learning, possesses distinctive characteristics compared to traditional feedforward neural networks. It incorporates feedback connections, enabling it to handle not only individual data points like images but also sequential data such as speech or video. Its applications range from unsegmented, connected handwriting recognition to speech recognition and anomaly detection in network traffic or intrusion detection systems (IDS). A typical LSTM unit consists of a cell along with an input gate, output gate, and forget gate. The cell retains information over varying time intervals, while the gates regulate the information flow in and out of the cell. LSTM networks excel in classifying, processing, and making predictions based on time series data, as they can account for unknown time lags between significant events. LSTMs were specifically developed to address the issue of vanishing gradients that can hinder training traditional RNNs. Another advantage of LSTMs is their relative insensitivity to gap length, making them favorable over RNNs, hidden Markov

models, and other sequence learning methods across numerous applications.

```
lstm_model = Sequential()
lstm_model.add(Embedding(len(tokenizer.word_index) + 1, 100, input_length=max_sequence_length))
lstm_model.add(LSTM(80))
lstm_model.add(Dense(1, activation='sigmoid'))

# Compile the model
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
lstm_model.summary()
```

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, 40, 100)	2653600
lstm (LSTM)	(None, 80)	57920
dense (Dense)	(None, 1)	81
=====		
Total params: 2,711,601		
Trainable params: 2,711,601		
Non-trainable params: 0		

After a simple LSTM-RNN architecture, we can improve the model by adding more information into it. One way to add more information to train upon is to provide word embeddings. These word embeddings carry a sense of context within them. Word embeddings generate multi-dimensional vector representations for words. The goal is to generate similar representations for words with similar meanings. In our next 2 models, we feed pretrained word embeddings into the LSTM architecture as the first embedding layer.

- Model 3 will input glove.6b.100d, GloVe is Stanford's unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
- Model 4 will input Word2vec embeddings. The effectiveness of Word2Vec comes from its ability to group together vectors of similar words. Given a large enough dataset, Word2Vec can make strong estimates about a word's meaning based on their occurrences in the text.

### Bidirectional LSTM with glove.6b.100d embeddings

```
bilstm_model_glove100 = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_len, weights=[embeddings_matrix], trainable=False),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    #return_sequences: will ensure output of first LSTM layer matches next
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
bilstm_model_glove100.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```



Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
embedding_1 (Embedding)	(None, 16, 100)	2653700
bidirectional_1 (Bidirectional)	(None, 16, 128)	84480
bidirectional_1_1 (Bidirectional)	(None, 64)	41216
flatten (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 6)	390
dropout (Dropout)	(None, 6)	0
dense_2 (Dense)	(None, 1)	7
=====		
Total params: 2,779,793		
Trainable params: 126,093		
Non-trainable params: 2,653,700		

## Gensim bidirectional LSTM + GRU model WITH Word2vec embedding model

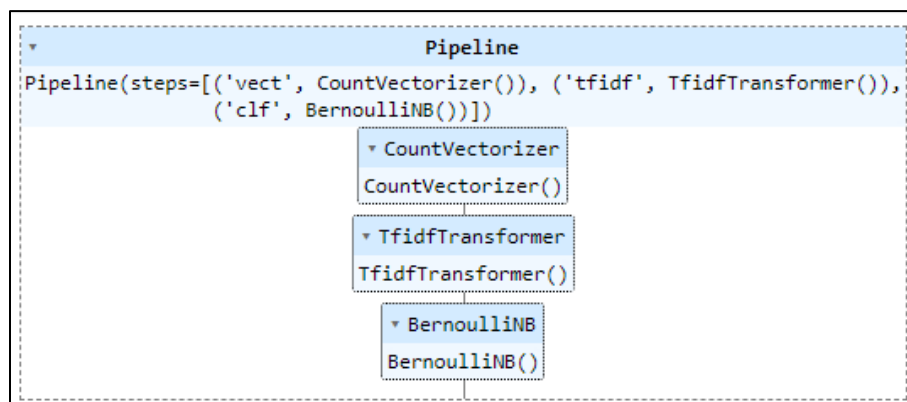
```

bilstmgru_model = Sequential()
#Non-trainable embedding layer
bilstmgru_model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM, weights=[embedding_vectors_w2v], input_length=20, trainable=True))
#LSTM
bilstmgru_model.add(Bidirectional(LSTM(units=128, recurrent_dropout = 0.3, dropout = 0.3, return_sequences = True)))
bilstmgru_model.add(Bidirectional(GRU(units=32, recurrent_dropout = 0.1, dropout = 0.1)))
bilstmgru_model.add(Dense(1, activation='sigmoid'))
bilstmgru_model.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.01), loss='binary_crossentropy', metrics=['acc'])

```

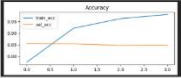
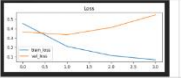
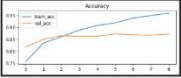
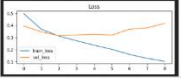
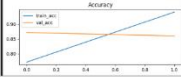
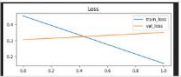
Model: "sequential_5"		
Layer (type)	Output Shape	Param #
=====		
embedding_4 (Embedding)	(None, 20, 200)	7320000
bidirectional_6 (Bidirectional)	(None, 20, 256)	336896
bidirectional_7 (Bidirectional)	(None, 64)	55680
dense_5 (Dense)	(None, 1)	65
=====		
Total params: 7,712,641		
Trainable params: 7,712,641		
Non-trainable params: 0		

## MODEL 5: NAÏVE BAYES





## Key findings and conclusions

MODEL	METRICS	ACCURACY	LOSS
1 REGRESSION	PRECISION - 0.83 RECALL - 0.82 F1 SCORE- 0.82 ACCURACY- 0.844		
2 LSTM-RNN	PRECISION - 0.80 RECALL - 0.88 F1 SCORE- 0.84 ACCURACY- 0.855		
3 BIDIRECTIONAL- LSTM	PRECISION - 0.88 RECALL - 0.82 F1 SCORE- 0.85 ACCURACY- 0.872		
4 NAIVE BAYES CLASSIFIER	PRECISION - 0.82 RECALL - 0.67 F1 SCORE- 0.74 ACCURACY- 0.78		
5 GENSIM BIDIRECTIONAL-LSTM	PRECISION - 0.87 RECALL - 0.83 F1 SCORE- 0.83 ACCURACY- 0.871		

Performance metrics of each model

From the above image, we can see that the baseline performance of our logistic regression with an accuracy of 0.844.

LSTM-RNN based architecture accuracy was 0.855. Although decreased in precision but significantly improved in recall. Precision can be interpreted as out of all the text that our model predicted as sarcastic, how many were actually sarcastic. While recall can be interpreted as out of total sarcastic texts in the dataset, how many were our model able to predict as sarcastic. So LSTM improved on recall i.e it was able to detect the sarcastic tweets more efficiently. This could be possible if the model overall classifies more number of labels as positive class thus decreasing precision but improving recall. This is a result of the RNN architecture that a lot of headlines are vaguely sarcastic or in the 'grey zone'. Identifying gray zone is the first step to handling the grey zone and our LSTM was able to achieve that.

But the key improvement came when the LSTM model was fed with pretrained word embeddings to learn context and semantics from. Bi-directional LSTM which was given Glove embedding had an accuracy of 0.872 while Gensim Bidirectional LSTM with Word2Vec embeddings returned a precision of 0.871. This is a much better performance (+0.05) over the baseline model. Hence showing evidence that the context of words is important in a sentence.

## ChatGPT Integration

The image below demonstrates calling the ChatGPT API to generate either a sarcastic or a non-sarcastic news headline. The returned headlines is passed as an input to our models and each of them predicts whether the input from GPT was sarcastic or not. We get good accuracy with LSTM models but logistic regression is not very accurate as seen below.

```
#!/pip install openai
import os
import openai

openai.api_key = "sk-y0eqA0SChv1JvIeX7lgYT3B1bkFJCiu1Cqk3ToFnYixsbdYO"

if (sarcasm == 1):
    message = "Asking ChatGPT for a sarcastic news headline."
    print(message)
else:
    message = "Asking ChatGPT for a non-sarcastic news headline."
    print(message)# Your code here

completion = openai.ChatCompletion.create(model="gpt-3.5-turbo",messages=[{"role": "user", "content": message}])
print(completion.choices[0].message.content)

run_model1(completion.choices[0].message.content)
run_model2(completion.choices[0].message.content)
run_model3(completion.choices[0].message.content)
run_model4(completion.choices[0].message.content)
```

Logistic Regression says the text is predicted as non-sarcastic  
3/3 [=====] - 0s 10ms/step  
LSTM-RNN Model says the text is predicted as sarcastic  
/usr/local/lib/python3.10/dist-packages/tensorflow/python/data/ops/structured\_function.py:254: UserWarning: Even though warnings.warn(  
3/3 [=====] - 0s 56ms/step  
LSTM Model with Glove Embeddings says the text is predicted as sarcastic  
167/167 [=====] - 110s 656ms/step  
LSTM-RNN Model says the text is predicted as sarcastic

The other major key findings are as follows.

- Data corpus size - The sarcastic detection requires large training dataset will also increases the resource required.
- Label quality - Quality of label data is very important and if the labeled data has the figurative of speech it will be useful in understanding what type of words and sentence will be used and it can also help in distinguish between different types of speech.
- ensuring no overfitting takes place - The model quite often leads to overfitting problem, especially if the quality of the label data is poor or training dataset is small.
- N gram – the application of n gram method in very important in sarcasm because while analysis the sentence in case sarcasm the meaning differs if it read as 3 or 4 words together compared with reading word by word analysis.
- Punctuation - The emotions might be expressed as part of punctuation, the meaning of a sentence can differ will a single punctuation eg:-
  - o Case 1: You look good.
  - o Case 2: You look good!
  - o From the above two cases adding a exclamatory gives a different sarcastic tone to the sentence.

## Further reading

There has actually been a new dataset published at the end of 2019, iSarcasm by Oprea and Magdy, where users contribute their own sarcastic tweets and include an explanation as to why it's sarcastic, as well as some metadata about them. Unfortunately it's not very big (around 1k tweets) but it's a small step in the right direction, in my opinion.

some people are going beyond text and using multimodal sarcasm detection with images and audio too, training the models on data from TV shows like The Big Bang Theory and Friends. It's still in the early stages but considering the multimodality of sarcasm, it looks promising. Many times the sarcasm is not in the text, but in the intonation or face expression.

- ❑ iSarcasm: A Dataset of Intended Sarcasm, Oprea et al., 2019 [[Paper](#)]
- ❑ CASCADE: Contextual Sarcasm Detection in Online Discussion Forums, Hazarika et al., 2018 [[Paper](#)]
- ❑ Sarcasm Detection: An In-depth Survey at Stanford University [[Paper](#)]
- ❑ Different Approaches in Sarcasm Detection A Survey : [[Paper](#)]
- ❑ Multi-Modal Sarcasm Detection in Twitter with Hierarchical Fusion Model, Cai et al., 2019 [[Paper](#)]
- ❑ Exploring Author Context for Detecting Intended vs Perceived Sarcasm, Oprea and Magdy, 2019 [[Paper](#)]
- ❑ Towards Multimodal Sarcasm Detection (An Obviously Perfect Paper), Castro et al., 2019 [[Paper](#)]
- ❑ Harnessing Sequence Labeling for Sarcasm Detection in Dialogue from TV Series 'Friends' [[Paper](#)]
- ❑ Challenges of Sarcasm Detection for Social Network: A Literature Review [[Paper](#)]