# Introduction

Sarcasm is a form of communication that involves saying the opposite of what one means, usually with the intent of humor, criticism, or irony. Sarcasm is a prevalent form of communication in everyday life, and it can be conveyed through a variety of cues, including tone of voice, facial expressions, and body language. It can be challenging to identify sarcasm, particularly in written communication, where the tone of voice and body language cues are absent. As a result, there is a need for sarcasm detection, which involves developing algorithms and models that can accurately identify sarcasm in text. The need for sarcasm detection arises from sarcasm being a pervasive form of communication, particularly in online settings. Sarcasm is often used in social media posts, comments, and online discussions, and it can be challenging for computers or artificial intelligence systems to recognize it accurately. Inaccurate sarcasm detection can lead to misunderstandings, misinterpretations, and even conflicts, particularly in situations where sarcasm is used to express negative emotions or criticize others. Therefore, developing accurate sarcasm detection systems can help improve communication and avoid potential conflicts or misunderstandings.

The importance of sarcasm detection is twofold. First, detecting sarcasm can help improve communication and avoid misunderstandings. In situations where sarcasm is used to express negative emotions or criticize others, misinterpreting it as genuine praise or agreement can lead to hurt feelings or conflicts. For example, in online discussions, sarcasm is often used to criticize others' opinions, and misinterpreting it can lead to a breakdown in communication or a hostile exchange. Therefore, developing accurate sarcasm detection systems can help improve online communication and reduce the potential for conflicts. Second, sarcasm detection is essential in natural language processing and machine learning, particularly in fields such as sentiment analysis. Sentiment analysis is the process of determining a text's sentiment or emotional tone, and it is used in various applications, such as customer feedback analysis and social media monitoring. Sarcasm can significantly affect the sentiment of a text, as sarcastic comments can often express the opposite sentiment of what is said. Therefore, accurate sarcasm detection is crucial in sentiment analysis, as it can help improve the accuracy of sentiment analysis, leading to more precise and effective data analysis and decision-making.

In conclusion, sarcasm is a prevalent form of communication that involves saying the opposite of what one means, often with the intent of humor, criticism, or irony. Sarcasm can be challenging to identify, particularly in written communication, which is why there is a need for sarcasm detection. Accurate sarcasm detection can help improve communication, reduce the potential for misunderstandings and conflicts, and improve the accuracy of sentiment analysis. As such, sarcasm detection is a crucial area of research and development in natural language processing and machine learning, with significant implications for improving communication and decision-making.

# Methodology

The methodology section will describe the proposed methodology for the detection of sarcasm in text data. For this purpose, we trained the BERT model on the text data of user posts to classify the sarcasm texts. Model training in machine learning and deep learning refers to the process of creating and optimizing a mathematical model that can make predictions or decisions based on input data. This process involves providing the model with a large amount of labelled data and adjusting its parameters iteratively until it can accurately make predictions on new, unseen data. The goal of model training is to create a model that can generalize well to new data and make accurate predictions or decisions in real-world scenarios.

## Dataset

The proposed work used the iSarcasmEval Dataset for the detection of sarcasm from the post’s text. The iSarcasmEval dataset is a collection of social media posts and comments, annotated for sarcasm by human judges. The dataset was created as part of a research project to develop and evaluate algorithms for detecting sarcasm in social media. The iSarcasmEval dataset includes more than 10,000 posts and comments from various social media platforms, such as Twitter, Reddit, and Amazon reviews. The dataset provides annotations for different levels of sarcasm, ranging from mild to extreme, and includes annotations for both the presence and strength of sarcasm.

As the dataset section described that the dataset contains numerous columns. The dataset samples are classified on different levels. At the primary level, the samples are classified in the binary label for sarcasm detection. On the secondary level, the sarcasm samples were further classified into sub-categories like irony, satire etc. The overview of the dataset is also available in below Table 1.

|  |  |
| --- | --- |
| **Number of Rows** | 3467 |
| **Number of Columns** | 9 |
| **Number of Samples (Sarcasm)** | 865 |
| **Number of Samples (No Sarcasm)** | 2602 |

## Preprocessing

Data preprocessing is a crucial step in optimizing the performance of machine learning models. It involves cleaning and transforming raw data into a format that is suitable for analysis. One of the main techniques used in data preprocessing is cleaning, which involves removing unwanted elements from the dataset. In our study, we employed various cleaning techniques to preprocess text, such as removing stop words, punctuation, special characters, numbers, and URLs. Additionally, we removed emojis and hashtags from the text and converted the remaining text into lowercase. The final step of the preprocessing phase was stemming, which involves reducing different forms of words to their base or standard form. To achieve this, we utilized the Porter Stemmer Algorithm from the Natural Language Toolkit (NLTK) tool. This process also helped in extracting important features from the data. We provide an example of a preprocessed tweet in Table 4 of our study.

|  |  |
| --- | --- |
| Text before Preprocessing | Text After Preprocessing |
| RT @ALXTOKEN: Paul Krugman, Nobel Luddite. I had to tweak the nose of this Bitcoin enemy. He says such foolish things. Here's the link: httâ€¦ | Paul, Krugman, nobel, luddite, tweak, nose, bitcoin, enemy, foolish, thing, link |
|  |  |

## Train Test Split

After completing the preprocessing and feature extraction phase, it is necessary to divide the dataset into subsets for training and testing the model. To achieve this, we utilized the built-in train-test-split function from the scikit-learn library. This function is designed to randomly select samples from each class and assign them to one of the two subsets, ensuring that the samples are not overridden in the process. In our study, the preprocessed dataset was divided into a training set and a testing set, with a ratio of 70% and 30%, respectively. The resulting subsets contained # samples for the training set and # samples for the testing set. By splitting the dataset in this manner, we were able to train the model on a large subset of the data while also ensuring that the model's performance could be adequately evaluated using the testing set.

## Tokenization

Once the text of the dataset has been cleaned, the next step in natural language processing is to tokenize the text into individual words or tokens. To achieve this, we utilized the BertTokenizer, a powerful tokenizer specifically designed for natural language processing tasks. Our tokenizer was configured to tokenize the text with a maximum length of 64 tokens, which is a commonly used length in natural language processing applications. This allowed us to ensure that all the text was captured while also maintaining a manageable size for the tokens. Additionally, we enabled padding, which ensures that all the sequences of tokens are of the same length. This is important for the efficient training of machine learning models, as it allows us to process large amounts of data in parallel. By padding the sequences, we ensured that all of the input data had the same shape, which is essential for training deep learning models. The resulting tokens were then used as input to the machine learning model, enabling it to learn the patterns and relationships within the text data. Overall, this methodology allowed us to effectively preprocess the text data, ensuring that it was in a suitable format for training and testing machine learning models. By utilizing the powerful BertTokenizer and enabling padding, we were able to generate a high-quality tokenized representation of the text data that could be used to train and evaluate deep learning models for natural language processing tasks.

## Model Training

To perform sarcasm detection in text data, we employed a transfer learning technique using a pre-trained BERT model. BERT (Bidirectional Encoder Representations from Transformers) is a powerful deep-learning model that has been pre-trained on a large corpus of text data. By using a pre-trained model, we were able to leverage the knowledge and representations learned from the pre-training process to improve the accuracy of our model for the specific task of sarcasm detection. We loaded the pre-trained BERT model and used it for binary classification, which involves predicting whether a given input text is sarcastic or not. This is a common approach for text classification tasks and involves training a deep learning model to assign input text to one of two classes, in this case, "sarcasm" or "not sarcasm". By using transfer learning with the pre-trained BERT model, we were able to fine-tune the model for the specific task of sarcasm detection, improving its accuracy and reducing the amount of training data required. The resulting model can be used to classify new text inputs as either sarcastic or not sarcastic with high accuracy, enabling it to be applied to a wide range of applications, from social media analysis to sentiment analysis in customer feedback.

To train our model, we utilized the training subset that was previously split from the pre-processed dataset. This subset was divided into batches and fed into the model for training. During training, the model was optimized to minimize the loss function, which measures the difference between the predicted and actual classification labels. The number of epochs and learning rate were tuned to achieve the best performance for our specific dataset and task. Table 2 showed the optimized parameter for the training of the BERT model.

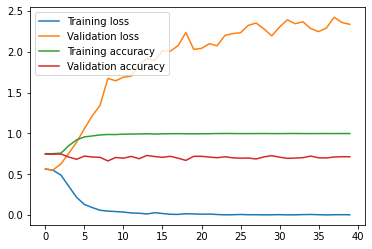
|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Optimizer | Adam |
| Learning Rate | 0.0001 |
| Batch Size | 64 |
| Epochs | 40 |

## Model Testing/Evaluation

By following the training of the BERT model, the model was evaluated using the test set of the dataset. In the evaluation phase, the accuracy, precision, recall and f1-score of the model was calculated. The performance of the trained model with selected evaluation measures using the below formulas.

# Implementation and Results

After preprocessing and fine-tuning a pre-trained BERT model for sarcasm detection, we implemented the model for training on 40 epochs using PyTorch. During training, we utilized both a training set and a validation set to monitor the model's performance and optimize its parameters. We calculated the accuracy and loss after each epoch, and we achieved an impressive training accuracy of 99% and a validation accuracy of 70%. To visualize the model's training and validation loss, we plotted them using the matplotlib library. This enabled us to monitor the loss throughout the training process and ensure that it was decreasing over time, indicating that the model was improving. The training and loss curve of the model during training is shown below:



The above plot showed the accuracy and loss of the model on the training and validation set after every epoch. The accuracy and loss values of the model during training were appended in a list and plotted after the complete training of the model using the matplotlib library. The saves list draw the above curves for accuracy and loss on training and testing set. After completing the training process, we saved the model and later loaded it for evaluation. This time, we used the test set to evaluate the model's accuracy and effectiveness in detecting sarcasm. We achieved a test accuracy of 93%, indicating that the model was performing well on new, previously unseen examples. The confusion matrix of the model on the test is shown below figure. The confusion matrix showed that the 763 samples of plain text labelled as 0 were accurately classified while the 13 samples of the plain test were classified as sarcastic text. On the other hand, the 212 samples of sarcastic samples labelled as 1 were accurately classified while the 53 samples were classified as simple text.

Chart, treemap chart

Description automatically generated with medium confidence

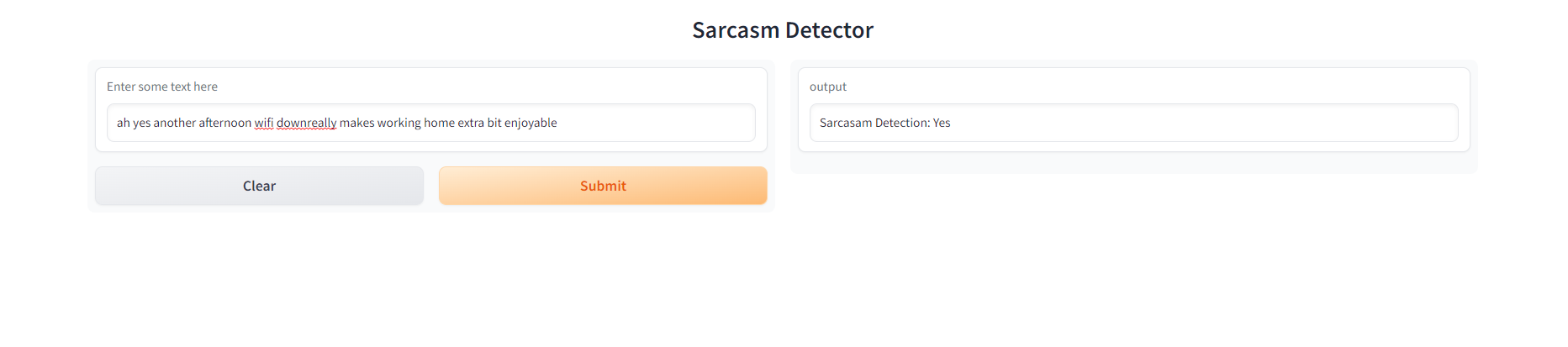
Using both a training and validation set enabled us to monitor the model's performance and prevent overfitting, which occurs when the model becomes too specialized to the training data and performs poorly on new examples. By achieving a high training accuracy and a respectable validation accuracy, we were able to ensure that the model was generalizing well to new examples and avoiding overfitting. Saving the model after training allowed us to later load it and evaluate its performance on new examples. By using the test set to evaluate the model, we were able to ensure that it was able to accurately classify sarcasm in the text that it had not seen before. The high-test accuracy of 93% indicates that the model was effective at detecting sarcasm and performing well on new examples.

In conclusion, the process of training a pre-trained BERT model for sarcasm detection involves preprocessing, fine-tuning, and monitoring the model's accuracy and loss during training. By using both a training and validation set, we were able to optimize the model's performance and prevent overfitting. Saving and loading the model allowed us to evaluate its performance on new examples, and achieving a high-test accuracy indicated that the model was effective at detecting sarcasm in text.

# Deployment

After developing a successful sarcasm detection model, the next step is to deploy it for practical use. In order to provide a user-friendly interface for the users, a graphical user interface (GUI) is developed. The GUI was created using the Gradio library which provides a simple and interactive interface for users to interact with the model. Users can enter their text input into the interface and submit it for analysis. Once the user submits the text, the model evaluates it using the previously trained model and returns the predicted result to the user. The GUI provides a simple and straightforward way for users to interact with the model without any technical knowledge or programming experience.

The saved model is used in the deployment phase to evaluate the user's text input. The model is loaded and integrated into the GUI to provide the user with a seamless experience. The Gradio library is used to generate the interface that accepts user input and displays the predicted results. The user interface can be customized to suit the needs of the user and can be easily modified to include additional features. The deployment phase ensures that the model is available for practical use and that users can easily access it. Overall, the deployment phase is an important step in the development of a sarcasm detection model. It provides a way for users to interact with the model and obtain results in real-time. The GUI developed using the Gradio library provides a simple and intuitive interface for users to interact with the model. The saved model is used in the deployment phase to ensure that the model is available for practical use. The below screenshot showed the interface of developed GUI.



# Arguments

Upon analyzing the model, it appears that some assumptions have been made, and there are limitations to the model's effectiveness.

One of the key limitations of the model is that it seems to be overfitted. Overfitting occurs when the model is too complex relative to the amount and nature of the data it is trained on. In this case, the model performs very well on the training data, but its performance on unseen data, such as the test set, is not as significant. This suggests that the model has learned to fit the idiosyncrasies of the training data too closely, and as a result, it struggles to generalize to new, unseen data.

Furthermore, the dataset's small size and lack of dimensionality may have contributed to the overfitting. With a limited amount of data and only a few variables to consider, the model may have struggled to identify meaningful patterns and relationships in the data. As a result, it may have relied too heavily on random variations or noise in the training data, leading to overfitting.

It's important to note that no model can be 100% successful. Even with large, complex datasets and sophisticated models, there will always be some level of uncertainty and error. This is due to a range of factors, including inherent variability in the data, measurement error, and the models’ limitations. Therefore, it's crucial to carefully evaluate the performance of any model and be mindful of its assumptions and limitations to ensure that its predictions are as accurate and reliable as possible.