Project Report:

Sarcasm and Irony Detection in

Natural Language Processing (NLP)

Submitted by

Ankit Makhija - 2023201021

Divyesh Patel - 2023201048

Md Hashmatur Rehman - 2021900012

---

1. Introduction

The detection of sarcasm and irony is an advanced task in Natural Language Processing (NLP), vital for accurate sentiment analysis and effective social media monitoring. Sarcasm detection significantly improves the understanding of real user sentiment on platforms like Twitter and Reddit, where sarcastic remarks are common. Traditional sentiment analysis models often struggle with this form of figurative language, requiring deeper contextual analysis.

Our project focuses on developing a robust sarcasm and irony detection model using deep learning techniques, including transformer architectures like BERT and RoBERTa, alongside contrastive learning approaches. Our goal is to enhance computational language models' ability to handle context-dependent sarcasm and irony, improving upon the current limitations of available models.

2. Problem Statement

Sarcasm and irony detection presents unique challenges due to:

- **Contextual Complexity:** Understanding sarcasm often requires more than text-based analysis, as tone and broader conversational context play a significant role.
- **Subtle Linguistic Cues:** Irony can be expressed through nuanced language patterns that are difficult for conventional NLP models to detect.
- **Cultural Variations:** Expressions of sarcasm differ across cultures, making model generalization difficult across datasets.

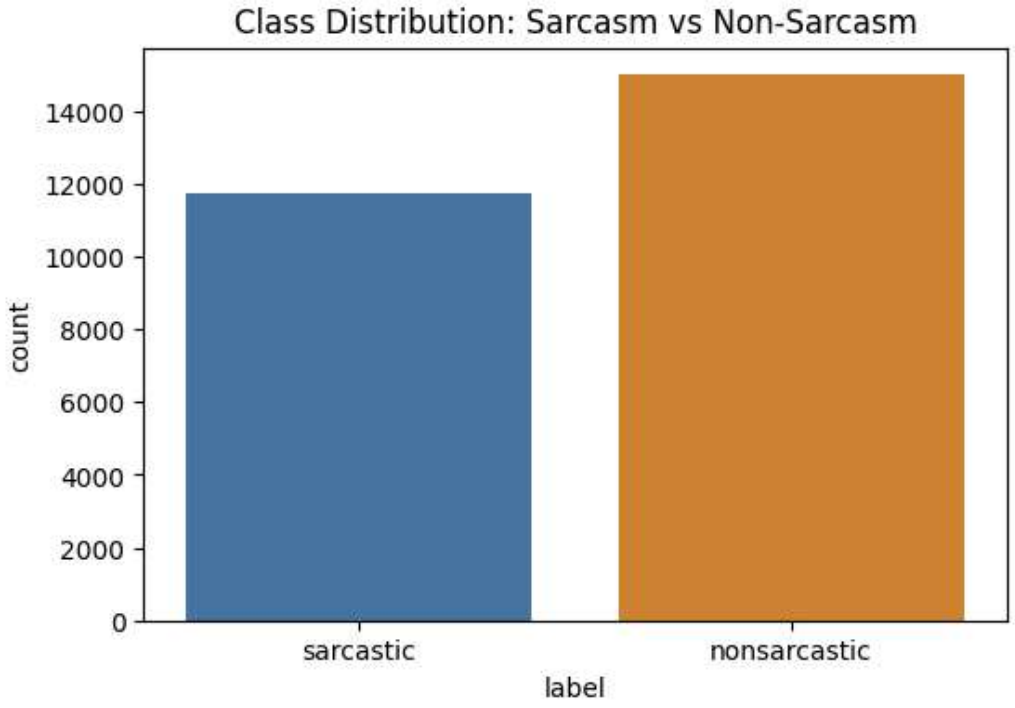
- **Data Scarcity:** Available labeled datasets are typically small, imbalanced, and lack sufficient context to train high-performing models.
- Our project aims to overcome these challenges using a transformer-based model that leverages contextual information and contrastive learning to differentiate between literal and sarcastic expressions.

3. Exploratory Data Analysis (EDA)

Dataset - The Headlines Dataset for Sarcasm Detection

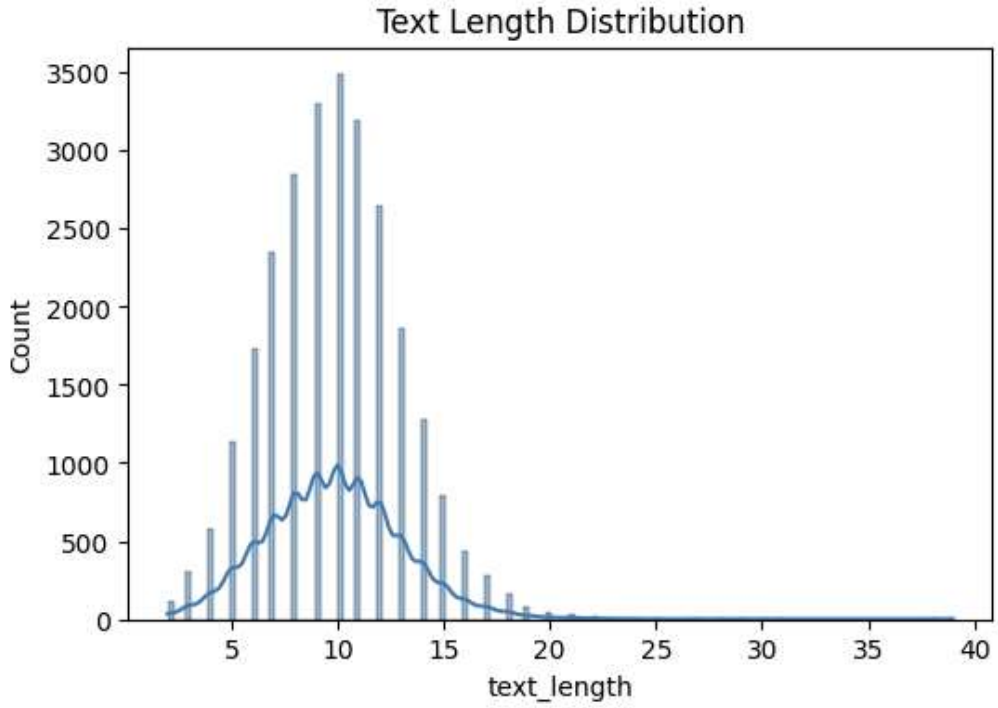
The Headlines Dataset for Sarcasm Detection is a collection of news headlines labeled for sarcasm, designed to help improve models for identifying sarcastic language in text. Each entry in the dataset contains a headline, a binary label indicating whether the headline is sarcastic (1) or non-sarcastic (0), and sometimes a link to the full article. With over 26,000 samples, this dataset presents a challenging task due to the subtle and context-dependent nature of sarcasm, making it difficult for traditional sentiment analysis or text classification models to accurately detect. It's commonly used for training machine learning models in NLP tasks like sarcasm detection, sentiment analysis, and text classification.

Class Distribution



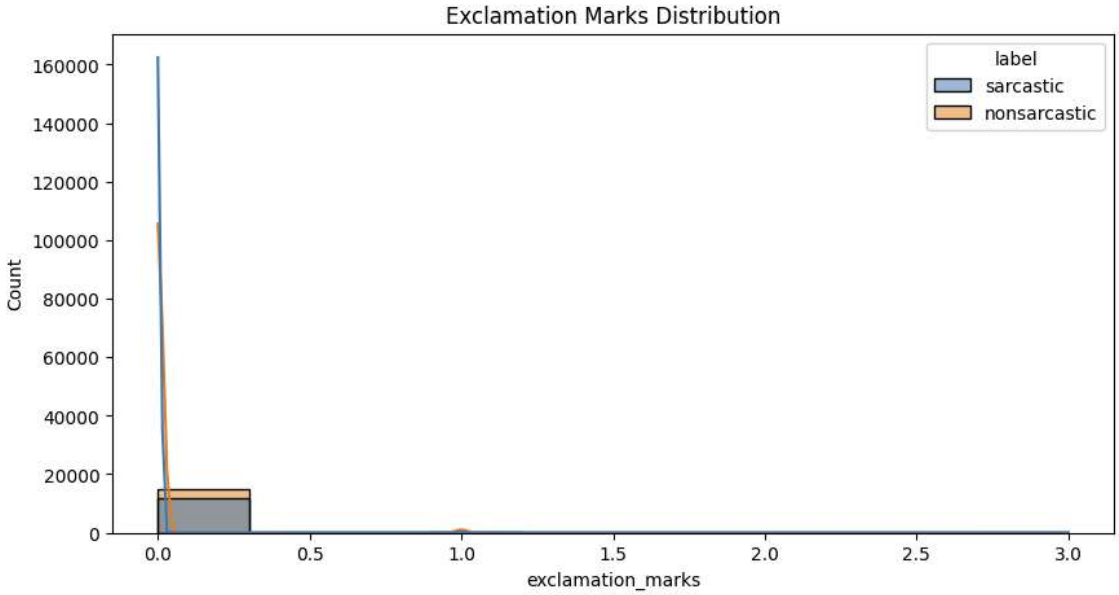
There is a slight imbalance in the dataset, with more non-sarcastic texts than sarcastic ones. This imbalance might require attention during model training, perhaps using class balancing techniques like oversampling or weighted loss functions.

Text Length Distribution



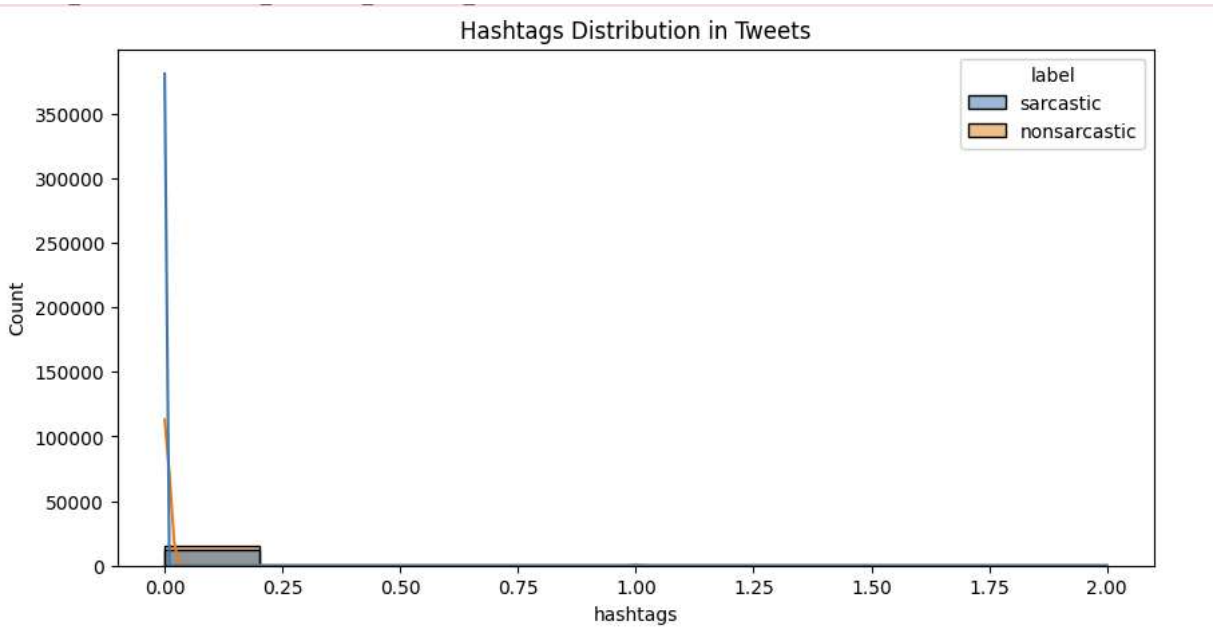
The distribution is relatively normal, with a peak around 8-15 words per text. There is no clear distinction between sarcastic and non-sarcastic texts based on length, though shorter texts (around 10 words) seem to dominate.

Exclamation Marks Distribution



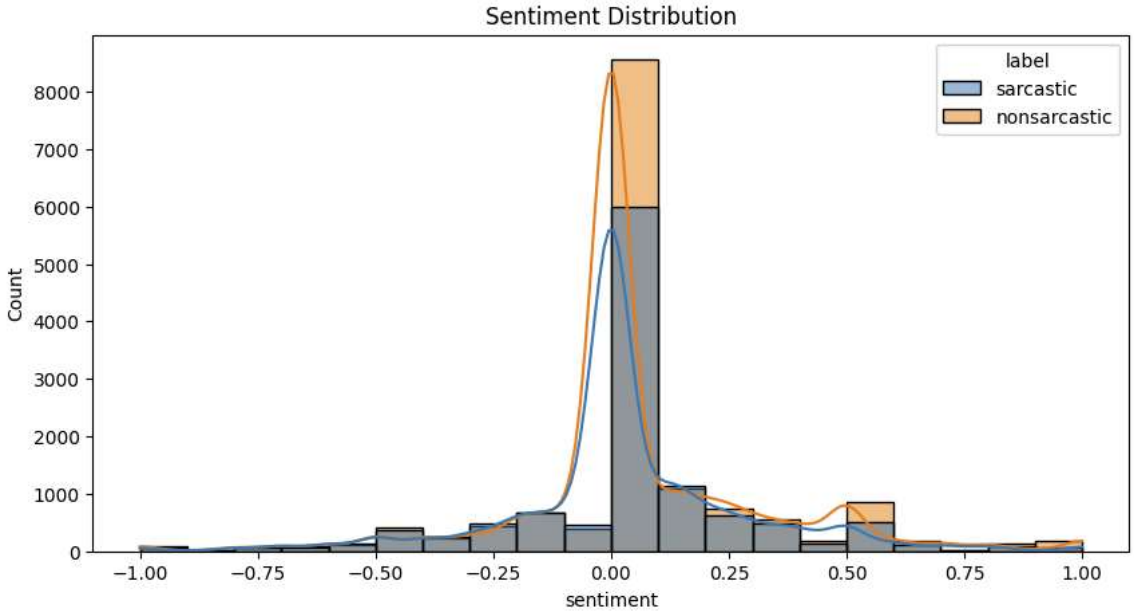
Sarcastic vs Non-Sarcastic: Exclamation marks are rarely used, and when they are, they tend to occur more frequently in sarcastic texts. This suggests that while rare, exclamation marks may be a useful indicator of sarcasm due to their exaggerated tone.

Hashtags Distribution



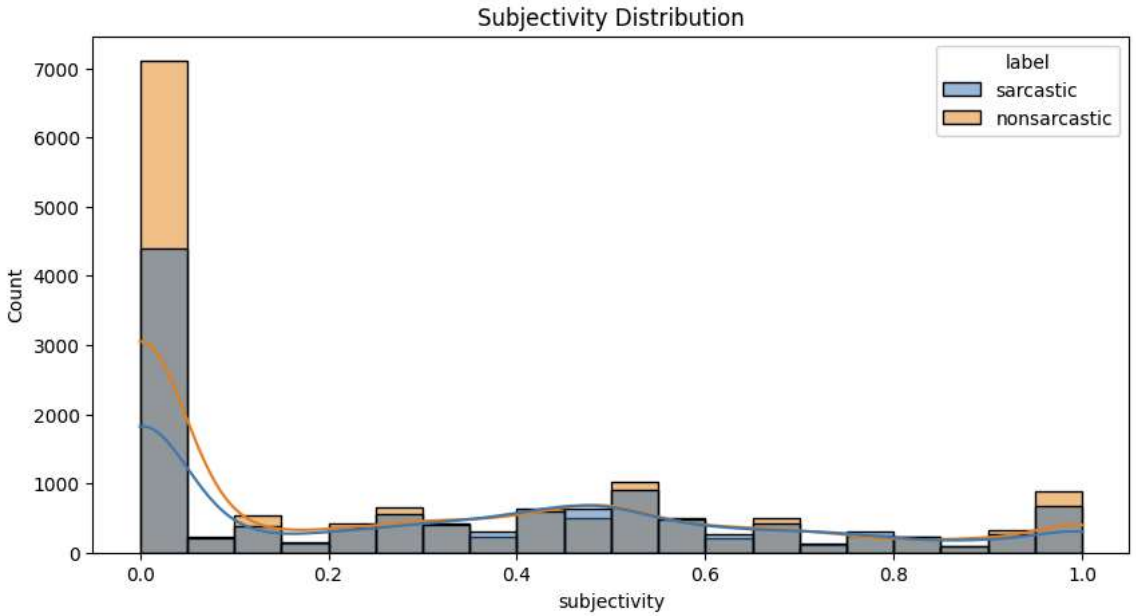
Sarcastic vs Non-Sarcastic: Both sarcastic and non-sarcastic texts show minimal use of hashtags. There is a significant peak at zero hashtags for both categories, suggesting that hashtags may not be a critical feature in detecting sarcasm for this dataset.

Sentiment Distribution



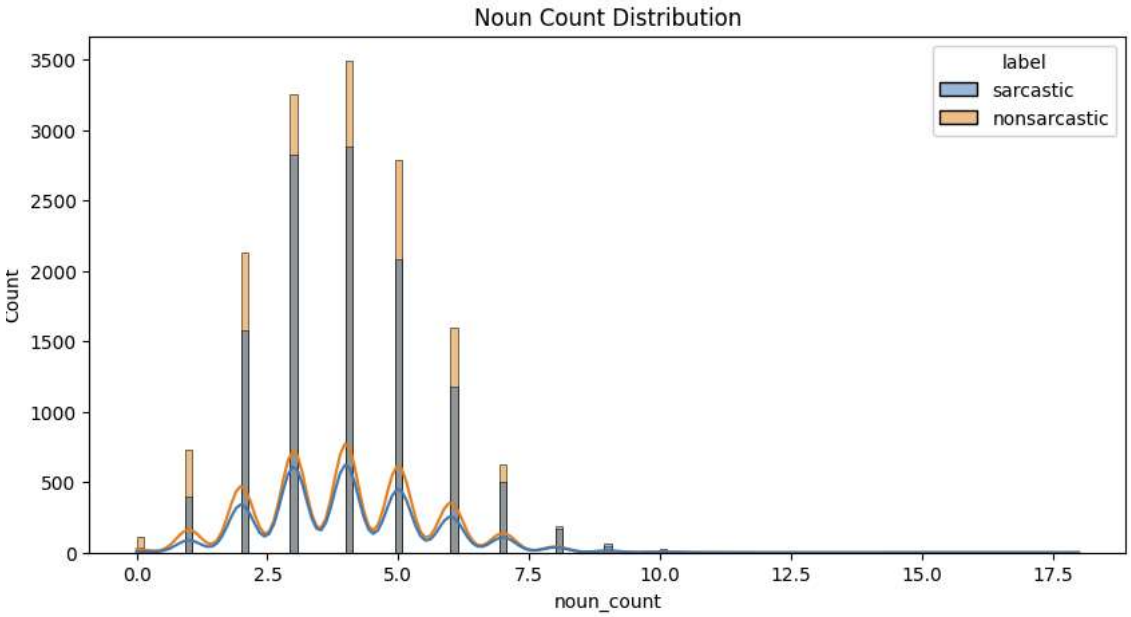
Sarcastic vs Non-Sarcastic: Both sarcastic and non-sarcastic texts have a neutral sentiment distribution (centered around 0), but sarcastic texts seem to have slightly more negative sentiments. This aligns with the idea that sarcasm often involves negative or contrasting sentiment despite surface-level positivity.

Subjectivity Distribution



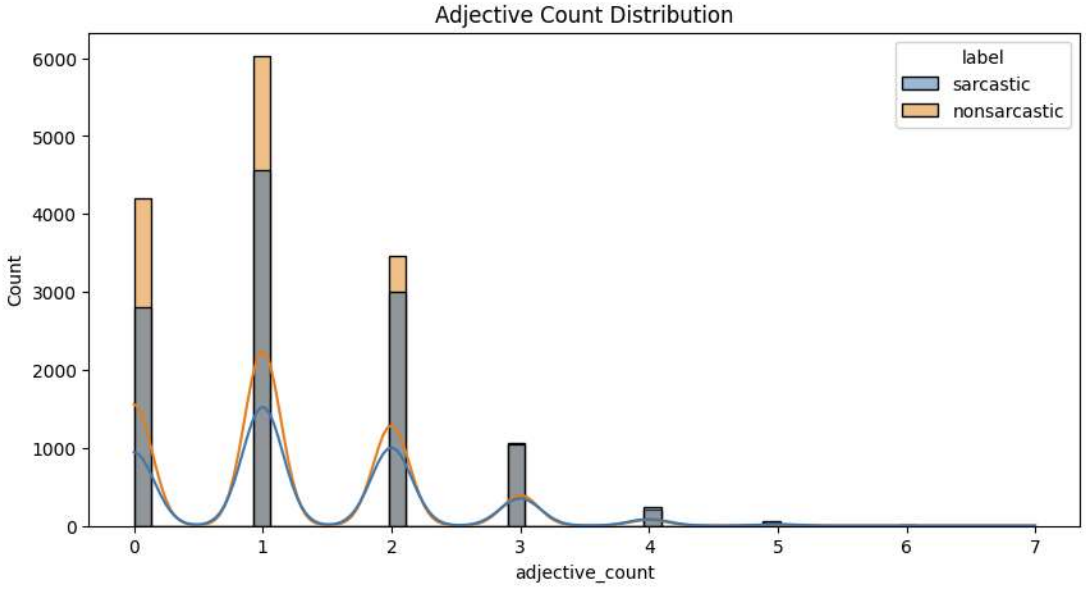
Sarcastic vs Non-Sarcastic: Non-sarcastic texts tend to have more texts with zero subjectivity (very objective), while sarcastic texts have a more spread-out subjectivity distribution. Sarcastic texts, by their nature, likely involve more personal opinions or subjective tones, which explains the higher variance.

Noun Count Distribution



Sarcastic vs Non-Sarcastic: Nouns are quite evenly distributed across sarcastic and non-sarcastic texts, with a peak around 3-5 nouns per text. The patterns are fairly similar, indicating that noun usage alone may not be a strong differentiator between sarcasm and non-sarcasm.

Adjective Count Distribution



Sarcastic vs Non-Sarcastic: Both sarcastic and non-sarcastic texts mostly use 0 or 1 adjective, but sarcastic texts seem to have slightly more instances of using two or three adjectives compared to non-sarcastic ones. This suggests that sarcastic texts may rely more on descriptive language to convey sarcasm or exaggeration.

Preprocessing

The preprocessing script performs a sequence of transformations to standardize and clean text data. Below is an overview of each transformation step:

---

1. Replace Emoticons

- **Function:** `replace_emoticons`
- **Description:** Identifies and replaces common emoticons or emojis with their textual descriptions (e.g., `:)` → `Smile`, `<3` → `heart`).
- **Purpose:** Simplifies emoticons into text form for better understanding by text-based models.

---

2. Replace Sarcasm Expressions

- Replaces patterns like `...` or `....` with the word `"possibility"`.
- **Purpose:** Handles ellipsis commonly used to express sarcasm or trailing thoughts.

---

3. Handle Numeric Patterns

- **Function:** Regex transformations
- Replaces various numeric patterns:
  - **Large numbers with commas** (e.g., `1,000`) → `NUM`
  - **Alphanumeric patterns** (e.g., `23pts`, `r800`) → `ALPHANUM`
  - **Ranges** (e.g., `5-10`) → `RANGE`
  - **Fractions** (e.g., `1/2`) → `fraction`
  - **Years** (e.g., `2003`) → `year`
  - **Amounts** (e.g., `500k`, `7million`) → `amount`
  - **Versions** (e.g., `4.0`) → `version`
  - **Standalone numbers** (e.g., `42`) → `NUM`

---

4. Time and Duration Patterns

- Replaces patterns related to time and duration:
  - **Times** (e.g., `3:30`, `1am`, `15minutes`) → `time`
  - **Durations** (e.g., `21st`, `200yrs`) → `duration`

---

5. Other Patterns

- Special Categories:
  - **Temperatures** (e.g., `100.20`) → `temperature`
  - **Speeds** (e.g., `90kph`) → `speed`
  - **Distances** (e.g., `155m`) → `distance`
- **URLs:** Matches web links (e.g., `http://example.com`) and replaces them with `URL`.

---

6. Token Replacement

- **Function:** `replace_token`
- Identifies tokens using regular expressions and replaces them with labels:
  - **Punctuations:** (`...`, `---`) → `punctuation`
  - **Special Characters:** (`@`, `%`, `*`) → `special`
  - **Currencies:** (`$`, `€`, `₹`) → `currency`

---

7. Hyphen Splitting

- Replaces hyphens (`-`) with spaces to split joined words or phrases.

---

8. Final Tokenization

- The text is tokenized by splitting on spaces.
- Each token is processed using `replace_token` for additional replacements.
- The tokens are joined back into a cleaned and normalized text.

Purpose of Preprocessing

This preprocessing pipeline is designed for:

- Standardizing text for natural language processing tasks.
- Normalizing emoticons, numeric data, and special characters into consistent labels.
- Making the text model-ready by removing noise and replacing specific patterns with generalized tokens.

Models Trained

Baseline Model (LSTM)

1. Overview

The model implemented is a Long Short-Term Memory (LSTM) neural network, designed for text classification. It processes tokenized sentences, embeds them into a high-dimensional space, and uses sequential learning to predict the labels of the input sequences.

---

2. Model Components

a. Input Layer

- **Input Size:** Vocabulary size (`len(vocab_dict)`).
- **Input Type:** Padded sequences of tokenized text, represented as integer indices.

b. Embedding Layer

- **Purpose:** Transforms token indices into dense vector representations.
- **Size:**
  - Input: `len(vocab_dict)` (vocabulary size).
  - Output: `hidden_size` (embedding dimension; set to 600).

c. LSTM Layer

- **Architecture:**
  - Single-layer LSTM.
  - **Hidden Size:** 600.
  - **Number of Layers:** 1.
  - **Batch First:** Enabled, ensuring batch dimension precedes sequence length.
- **Purpose:** Captures temporal dependencies and contextual relationships in the sequence.

d. Fully Connected Layer

- **Input:** Final hidden state of the LSTM for each sequence.
- **Output:** Two logits corresponding to the two classes (sarcastic and nonsarcastic).
- **Size:**
  - Input: 600 (hidden size of LSTM).
  - Output: 2 (number of classes).

e. Activation Function

- **Purpose:** Outputs raw logits for the classification task.
- No explicit activation is applied at the final layer because the CrossEntropyLoss function expects raw logits.

---

3. Hyperparameters

- **Hidden Size:** 600.
- **Embedding Dimension:** 600.
- **Number of Layers:** 1.
- **Learning Rate:** 0.001.
- **Batch Size:** 128.
- **Number of Epochs:** 6.

---

4. Loss Function

- **Type:** CrossEntropyLoss.
- **Purpose:** Computes the categorical loss between predicted logits and true labels.

---

5. Optimization Algorithm

- **Type:** Adam Optimizer.
- **Learning Rate:** 0.001.
- **Purpose:** Minimizes the loss function and updates the weights in the model.

BERT

Model Architecture

1. Base Model

- **Model Type**: BERT (Bidirectional Encoder Representations from Transformers)
- **Pretrained Version**: bert-base-uncased
- **Purpose**: Fine-tuned for binary sequence classification (sarcasm detection in text).
- **Input**:
    - Preprocessed and tokenized text data.
    - Maximum sequence length: 128 tokens.
    - Includes input_ids and attention_mask.
- **Output**:
    - Classification logits for two classes (sarcastic and non-sarcastic).

2. Model Components

- **Embedding Layer**:
    - Maps input tokens to dense vectors of size 768 (hidden size).
    - Incorporates positional embeddings for word order.
- **Encoder Layers**:
    - 12 transformer layers with self-attention mechanisms.
    - Attention heads per layer: 12.
    - Hidden size: 768.
- **Classification Head**:
    - Fully connected feed-forward layer with softmax activation.
    - Maps the pooled output from BERT's [CLS] token to logits for binary classification.

3. Tokenization

- **Tokenizer**: BertTokenizer (from bert-base-uncased).
- **Key Features**:
    - Converts input text to token IDs.
    - Pads sequences to a maximum length of 128.
    - Truncates sequences longer than 128 tokens.

Hyperparameters

- **Batch Size**: 16 (for both training and validation).
- **Learning Rate**: 2e-6 (optimized with AdamW).
- **Optimizer**: AdamW (decoupled weight decay).

- **Epochs**: Maximum of 8 with early stopping (patience = 3).
- **Loss Function**: Cross-entropy loss for binary classification.
- **Device**: Utilizes GPU (cuda) if available, else defaults to CPU.

Training Details

- **Training Strategy**:
    - Fine-tuning the pre-trained BERT model.
    - Includes updating all layers of BERT.
- **Validation**:
    - Performance monitored using validation loss.
    - Best model weights saved based on lowest validation loss.
- **Early Stopping**:
    - Stops training if validation loss does not improve for 3 consecutive epochs.

Roberta

1. Model Overview

The model is based on **CardiffNLP's Twitter RoBERTa-base Sentiment** transformer, fine-tuned for a binary sarcasm detection task. The transformer architecture leverages pre-trained embeddings and self-attention mechanisms to capture nuanced semantic and syntactic relationships in text data.

---

2. Pretrained Model

a. Backbone

- **Model Name:** `cardiffnlp/twitter-roberta-base-sentiment`
- **Architecture:**
  - RoBERTa (Robustly Optimized BERT Pretraining Approach).
  - Transformer-based model with self-attention mechanisms.
  - Pretrained on large-scale Twitter sentiment datasets, optimized for short and informal text.

b. Modifications

- The classification head of the pretrained model is replaced to match the requirements of this task:
  - **Output Layer:** A fully connected layer with two output logits for binary classification (sarcastic and non-sarcastic).

---

3. Input Representation

a. Tokenization

- **Tokenizer Used:** `AutoTokenizer` from Hugging Face.
- **Method:**
  - Tokenizes text into subword tokens.
  - Adds special tokens (e.g., `[CLS]`, `[SEP]`) required by the RoBERTa model.
  - Pads/truncates sequences to a maximum length of 128 tokens.
  - Generates:
    - `input_ids`: Encoded tokens.
    - `attention_mask`: Mask to differentiate padding tokens from real tokens.

b. Dataset

- **Training Data:**
  - Tokenized text data, with preprocessed headlines or tweets.
  - Labels converted into binary format (`1` for sarcastic, `0` for non-sarcastic).
- **Custom Dataset Class:**
  - Implements the PyTorch Dataset interface to handle tokenized inputs (`input_ids`, `attention_mask`) and corresponding labels.

---

4. Fine-Tuning Approach

a. Loss Function

- **Type:** Weighted CrossEntropyLoss.
- **Details:**
  - The class imbalance in the dataset is addressed using weighted loss:
    - Weights calculated as the inverse proportion of class occurrences.
    - Ensures the model does not overly favor the majority class.

b. Training Strategy

- **Trainer Class:** Hugging Face `Trainer` with a custom `compute_loss` method to integrate weighted loss.
- **Optimization Algorithm:** AdamW (implemented within Hugging Face's Trainer).
- **Hyperparameters:**
  - **Learning Rate:** Determined through grid search.
  - **Batch Size:** Tuned via grid search.
  - **Warmup Steps:** Gradual increase of learning rate during early iterations.
  - **Weight Decay:** Regularization to avoid overfitting.

c. Hyperparameter Grid Search

- **Parameters Explored:**
  - Learning Rate: [1e-6, 5e-6, 1e-5].
  - Batch Size: [16, 32].
  - Weight Decay: [1e-4, 1e-2].
  - Warmup Steps: [100, 500].
  - Number of Epochs: [2, 3].
- **Metric for Selection:** F1 Score (computed on validation data).

---

5. Evaluation Metrics

- **Accuracy:** Percentage of correct predictions.
- **F1 Score:** Harmonic mean of precision and recall.
- **Confusion Matrix:** Provides insights into misclassifications.
- **Report:** Precision, recall, and F1-score for each class.

---

6. Final Model Training

a. Best Parameters

- Identified through grid search and used for the final training of the model.

b. Output

- The model and tokenizer are saved for future use (`./sarcasm_detector_model_tuned`).

---

7. Advantages of This Architecture

- **Pretrained Features:** Utilizes a robust model already trained on a similar domain (Twitter).
- **Weighted Loss:** Tackles class imbalance effectively.
- **Fine-Tuning:** Adapts pretrained weights to the sarcasm detection task with minimal overhead.
- **Custom Trainer:** Extensible and integrates seamlessly with the Hugging Face ecosystem.

RNN Roberta

1. Overview

The model combines the strengths of the **RoBERTa-base transformer** for pre-trained language understanding with a **Recurrent Convolutional Neural Network (RCNN)** to enhance context capture and sequence modeling. It is designed for binary sarcasm detection in text data.

---

2. Key Components

a. Pretrained RoBERTa

- **Model Name:** `roberta-base`.
- **Purpose:** Provides contextualized embeddings for the input text using self-attention and pretraining on large corpora.
- **Output:** Hidden states (`last_hidden_state`) for each token in the input sequence.

b. Bidirectional LSTM

- **Input:** Hidden states from RoBERTa (`last_hidden_state`).
- **Configuration:**
  - Hidden Size: 64 (bidirectional).
  - Layers: 1.
  - Bidirectional: True.
- **Purpose:** Captures temporal dependencies in both forward and backward directions.

c. Feature Fusion Layer

- **Input:** Concatenation of:
  - Hidden states from RoBERTa.
  - Outputs from the bidirectional LSTM.
- **Transformation:** Linear layer (`W`) maps the concatenated features back to the original hidden size of RoBERTa.

d. Max Pooling

- **Purpose:** Extracts the most salient features across the sequence.
- **Output:** Single pooled feature vector for classification.

e. Dropout

- **Rate:** 0.1.
- **Purpose:** Regularizes the model and prevents overfitting.

f. Fully Connected Layer

- **Input:** Pooled features.
- **Output:** Logits for two classes (sarcastic, non-sarcastic).

---

3. Training Configuration

- **Loss Function:** CrossEntropyLoss.
- **Optimizer:** AdamW.
- **Learning Rate:** 2e-5.
- **Batch Size:** 16.
- **Weight Decay:** 1e-5.
- **Number of Epochs:** 5.

---

4. Advantages

- **Pretrained RoBERTa:** Leverages state-of-the-art embeddings for contextual understanding.
- **RCNN Fusion:** Enhances feature extraction with bidirectional LSTMs and pooling.
- **Robust Regularization:** Dropout and weight decay help mitigate overfitting.

Results

Model	Accuracy	Precision	Recall	F1- score
BaseLine(LSTM)	0.8136	0.8221	0.8395	0.8301
BERT	0.9199	0.8918	0.9218	0.9066
Roberta	0.8723	0.8523	0.8673	0.8597
RNN Roberta	0.94	0.94	0.92	0.93

Cross-Linguistic Transfer for Sarcasm Detection Using BERT

Objective

Subsequently, on sarcasm detection using BERT, applying transfer learning strategies to address the challenge of limited data in the target language. By using a larger, semantically similar dataset, the aim is to create a model that balances generalization with language-specific adaptation.

Proposed Methodology

To enable cross linguistic sarcasm detection , our approach involves three training strategies:

1. **Direct Fine-Tuning**:
    - The model is fine-tuned directly on the target dataset.
    - Pros: Focuses entirely on the nuances of the target domain.
    - Cons: Limited data increases the risk of overfitting and poor generalization.
2. **Cross-Domain Transfer**:
    - The model is pre-trained on a large dataset from a different domain (e.g., English sarcasm data) and evaluated directly on the target dataset.
    - Pros: Utilizes a large dataset to capture general sarcasm-related features.
    - Cons: Lacks adaptation to the specific linguistic and stylistic traits of the target domain.
3. **Sequential Fine-Tuning**:
    - The model is first fine-tuned on the large dataset to learn general sarcasm detection features.
    - It is then fine-tuned on the smaller target dataset to adapt to its unique characteristics.
    - Pros: Combines the strengths of generalization and domain-specific learning, reducing overfitting while adapting to the target dataset.

Rationale

The sequential fine-tuning strategy is designed to:

- **Extract General Features**: The initial phase captures sarcasm-related cues (e.g., sentiment and contextual nuances) from a larger dataset.
- **Adapt to the Target Domain**: The second phase refines these features to align with the linguistic and stylistic patterns in the target dataset.
- **Reduce Overfitting**: Training on a larger dataset regularizes the model, improving its ability to generalize to unseen data.

Results

1. **Direct Fine-Tuning**:

Hyperparameters			Accuracy
Epochs	**Learning Rate**	**Patience**	
8	1e-5	3	0.8854
10	2e-4	3	0.8958
12	2e-5	3	0.8750
15	2e-5	5	0.8958

Best Accuracy: 0.8958

2. **Cross-Domain Transfer**:

Hyperparameters			Accuracy
Epochs	**Learning Rate**	**Patience**	
8	2e-5	3	0.8333
12	2e-6	3	0.8437

Best Accuracy: 0.8437

3. **Sequential Fine-Tuning**:

Hyperparameters			Accuracy
Epochs	**Learning Rate**	**Patience**	
12	2e-6	3	0.8958

Best Accuracy: 0.8958

Conclusion

This approach highlights the potential of transfer learning in low-resource settings. Sequential fine-tuning, in particular, demonstrates how generalisable features can be effectively adapted for specific linguistic and cultural nuances, enabling robust sarcasm detection in multilingual contexts.