



Politechnika Krakowska
im. Tadeusza Kościuszki

**JSC “Tadeusz Kościuszko Kraków University of Technology”
Faculty of Computer Science and Telecommunications**

Natural language processing (NLP) Sarcasm Detector Project Report

Performed by:

Danel Kanbakova,

Zhanel Aldan,

Aruzhan Satybaldy

Teachers:

Radosław Kycia

Krakow, 2025

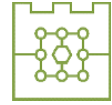


Table of content:

1. Abstact	
2. Introduction	
2.1 Aim	
2.2 Scope	
2.3 Methodology	
3. Theoretical Part	
3.1 Supervised Learning Algorithms	
3.2 Neural Network Models	
3.3 Transformer-based Models	
4. Practical Part	
4.1 Data Collection	
4.2 Data Preprocessing	
4.3 Model Implementation	
4.4 Model Evaluation	
5. Summary	
6. References	

Sarcasm Detector Project Report

1. Abstract

Detecting sarcasm in text is a challenging natural language processing task with applications in sentiment analysis and human-computer interaction. This project aims to develop and compare machine learning and deep learning models for sarcasm detection using a balanced Kaggle dataset. The workflow includes data exploration, preprocessing, feature extraction, model training, and evaluation. Transformer-based models demonstrated superior performance compared to traditional and recurrent neural network approaches.

2. Introduction

2.1 Aim

The main objective is to design and evaluate several models capable of identifying sarcasm in text data, thereby improving automated understanding of nuanced language.

2.2 Scope

The project covers dataset acquisition, text preprocessing, model building with supervised learning, neural networks, and transformers, as well as model assessment using appropriate metrics.

2.3 Methodology

Implemented in Python using Jupyter Notebook and libraries including Pandas, NLTK, Scikit-learn, TensorFlow/Keras, and Hugging Face Transformers. The approach involves cleaning raw text, extracting features, training diverse models, and rigorous evaluation.

3. Theoretical Part

In this project, we explored multiple modeling approaches for sarcasm detection, each bringing distinct strengths and challenges.

3.1 Supervised Learning Algorithms

Logistic Regression is a linear model used for binary classification that models the log-odds of the target class as a linear combination of input features. Its simplicity and interpretability make it a common baseline in NLP tasks. Despite its linear nature, with appropriate feature engineering (such as TF-IDF), it can capture important textual signals related to sarcasm. It requires relatively little computational power and trains quickly, making it a practical choice for large datasets.

Gradient Boosting Machines (GBM), specifically implementations like XGBoost or LightGBM, build an ensemble of decision trees iteratively, where each new tree corrects errors made by previous ones. This method excels in capturing complex, nonlinear relationships in data, such as the subtle contextual cues of sarcasm. Gradient Boosting models also provide feature importance scores, which can help interpret which words or patterns contribute most to sarcasm detection. Regularization parameters and tree depth control allow for mitigation of overfitting, which is critical in dealing with noisy text data.

3.2 Neural Network Models

Long Short-Term Memory (LSTM) networks are designed to solve the vanishing gradient problem inherent in standard RNNs, enabling them to remember information over longer sequences. This capability is essential in sarcasm detection, where the meaning can depend on the relationship between words far apart in a sentence or paragraph. Our LSTM architecture included an embedding layer that transforms words into dense vector representations, capturing semantic similarity. Dropout layers were added to reduce overfitting, which is common in deep networks trained on limited samples.

Gated Recurrent Units (GRU) offer a simplified alternative to LSTMs with fewer parameters, which often translates to faster training and less risk of overfitting. GRUs combine the forget and input gates into a single update gate, maintaining the ability to capture dependencies in sequences effectively. Given resource constraints, GRUs can be a practical choice when training time or dataset size is limited, while still capturing temporal dynamics in text.

3.2 Transformer-based Models

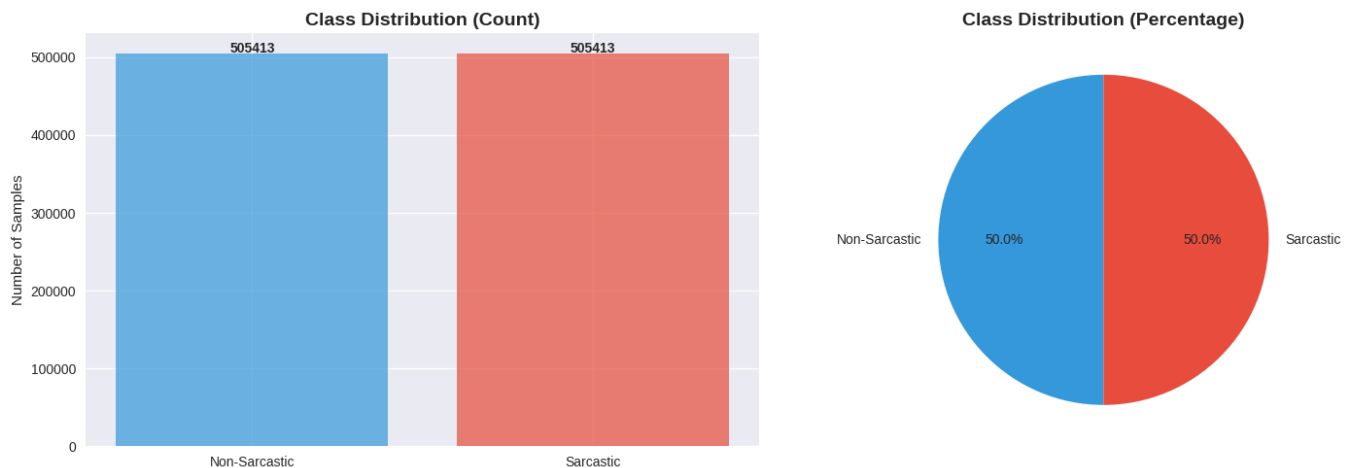
RoBERTa (Robustly optimized BERT approach) is an advancement over the original BERT model. It is pretrained on significantly larger datasets with modified training procedures such as dynamic masking and removal of the next sentence prediction task, which improves its generalization capabilities. Transformers use self-attention mechanisms that allow the model to weigh the importance of different words contextually across the entire input sequence. This is particularly advantageous for sarcasm detection, where meaning depends on subtle cues and context that may be non-local. RoBERTa's architecture, combined with fine-tuning on task-specific data, enables state-of-the-art performance in understanding nuanced language.

4. Practical Part

Exploratory data analysis included word clouds for sarcastic and non-sarcastic comments, comment length distributions, and subreddit analysis to understand sarcasm distribution.

4.1 Data Collection

The dataset used in this project was sourced from Kaggle and contains approximately 1.3 million comments labeled as sarcastic or non-sarcastic. The data was collected from Reddit and other social media platforms, providing a rich, diverse corpus of natural language with real-world examples of sarcasm. The dataset is balanced, which is crucial to prevent model bias toward the majority class and to ensure the model learns to distinguish sarcasm effectively.



4.2 Data Preprocessing

Effective preprocessing is critical for natural language tasks:

- **Text Cleaning:** We removed digits, punctuation, and converted all text to lowercase to reduce noise.

```
def advanced_preprocess(text, remove_stopwords=True, lemmatize=True):  
    """  
    Advanced preprocessing with tokenization, stopwords removal, and lemmatization  
    """  
    # Clean text  
    text = clean_text(text)
```

- **Stopword Removal:** Commonly used words with little semantic value (e.g., "the," "is") were removed to focus on informative tokens.

```
# Remove stopwords
if remove_stopwords:
    tokens = [token for token in tokens if token not in stop_words]
```

- **Tokenization:** Text was split into tokens (words or subwords), the fundamental units for model input. For traditional ML models, we used word-level tokenization; for transformer models, subword tokenization with Byte-Pair Encoding was employed.

```
# Tokenize
tokens = word_tokenize(text)
```

- **Vectorization:** For Logistic Regression and Gradient Boosting, TF-IDF vectorization transformed tokens into numerical features reflecting word importance within documents and across the corpus.
- **Sequence Preparation:** For LSTM and GRU, sequences were converted into integer indices representing tokens, then padded to uniform length to accommodate batch processing.
- **Tokenizer Usage:** The Hugging Face tokenizer tailored for RoBERTa handles tokenization, padding, and encoding in a way optimized for the transformer model.

Additionally, exploratory data analysis included:

- **Word Clouds:** Visualizing the most frequent words in sarcastic and non-sarcastic classes, revealing distinct lexical patterns.

```
# Create word clouds for sarcastic and non-sarcastic comments
fig, axes = plt.subplots(1, 2, figsize=(20, 8))

# Sarcastic comments word cloud
sarcastic_text = ' '.join(df[df['label'] == 1]['processed_comment'].tolist())
wordcloud_sarcastic = WordCloud(width=800, height=400, background_color='white',
                                colormap='Reds', max_words=100).generate(sarcastic_text)

axes[0].imshow(wordcloud_sarcastic, interpolation='bilinear')
axes[0].set_title('Sarcastic Comments - Word Cloud', fontsize=16, fontweight='bold')
axes[0].axis('off')

# Non-sarcastic comments word cloud
non_sarcastic_text = ' '.join(df[df['label'] == 0]['processed_comment'].tolist())
wordcloud_non_sarcastic = WordCloud(width=800, height=400, background_color='white',
                                    colormap='Blues', max_words=100).generate(non_sarcastic_text)

axes[1].imshow(wordcloud_non_sarcastic, interpolation='bilinear')
axes[1].set_title('Non-Sarcastic Comments - Word Cloud', fontsize=16, fontweight='bold')
axes[1].axis('off')

plt.tight_layout()
plt.show()
```



- **Comment Length Analysis:** Statistical analysis showed sarcastic comments tend to be longer on average, providing a potential discriminative feature.

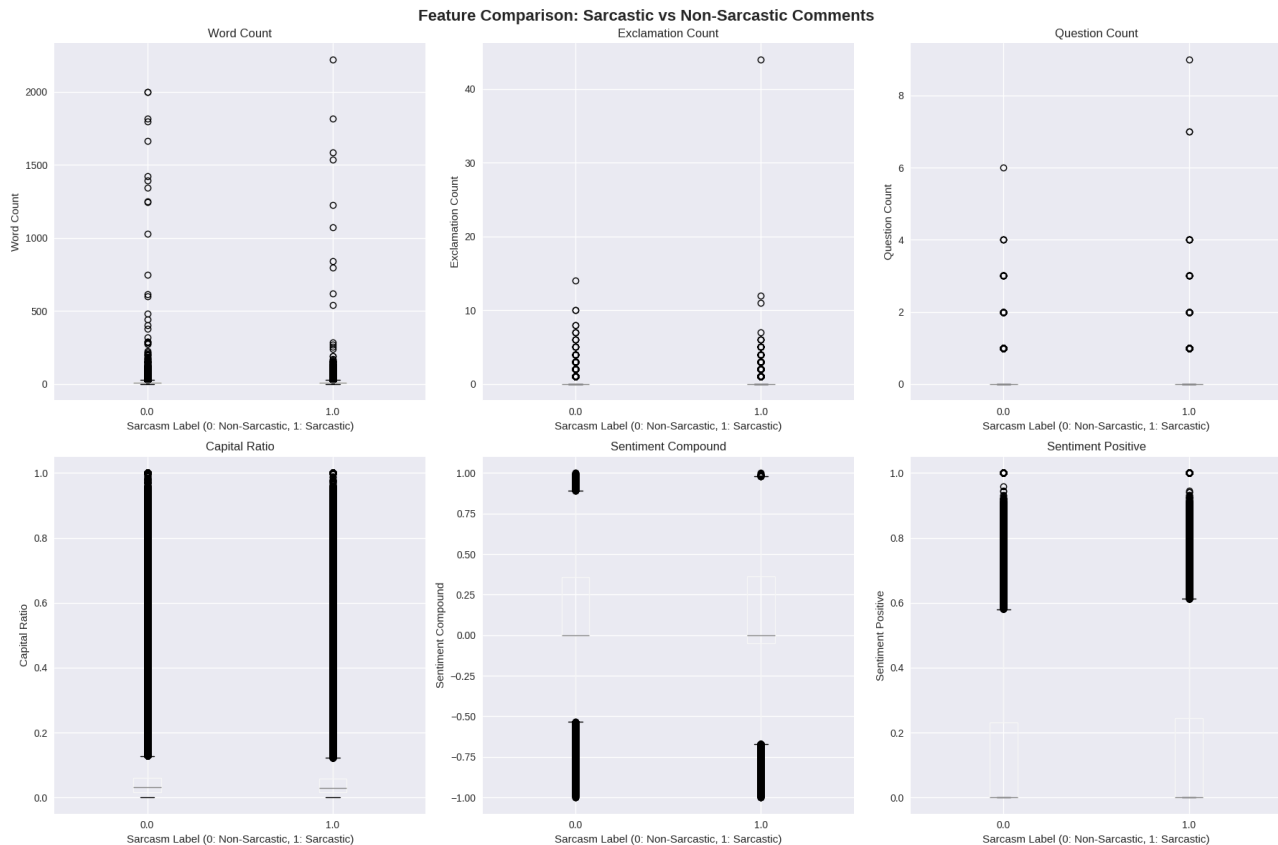
```
[ ] # Feature comparison between sarcastic and non-sarcastic comments
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# Define features to compare
features_to_compare = ['word_count', 'exclamation_count', 'question_count',
                      'capital_ratio', 'sentiment_compound', 'sentiment_positive']

for i, feature in enumerate(features_to_compare):
    row = i // 3
    col = i % 3

    # Create box plot
    df_features.boxplot(column=feature, by='label', ax=axes[row, col])
    axes[row, col].set_title(f'{feature.replace("_", " ").title()}')
    axes[row, col].set_xlabel('Sarcasm Label (0: Non-Sarcastic, 1: Sarcastic)')
    axes[row, col].set_ylabel(feature.replace('_', ' ').title())

plt.suptitle('Feature Comparison: Sarcastic vs Non-Sarcastic Comments',
            fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```



- **Subreddit Analysis:** Identification of subreddits with higher sarcasm rates helped understand the context and style variations in sarcastic language.

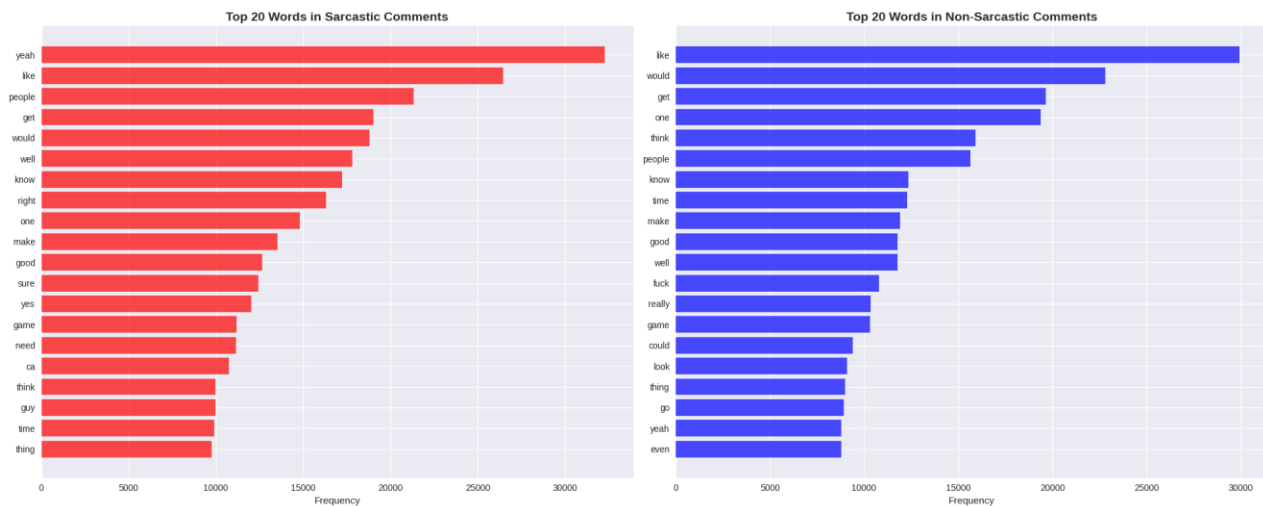

```
def get_top_words(text_list, n=20):
    """
    Get top n words from a list of texts
    """
    all_words = ' '.join(text_list).split()
    return Counter(all_words).most_common(n)

# Get top words for each class
sarcastic_words = get_top_words(df[df['label'] == 1]['processed_comment'].tolist())
non_sarcastic_words = get_top_words(df[df['label'] == 0]['processed_comment'].tolist())

# Plot top words
fig, axes = plt.subplots(1, 2, figsize=(20, 8))

# Sarcastic words
words, counts = zip(*sarcastic_words)
axes[0].barh(range(len(words)), counts, color='red', alpha=0.7)
axes[0].set_yticks(range(len(words)))
axes[0].set_yticklabels(words)
axes[0].set_title('Top 20 Words in Sarcastic Comments', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Frequency')
axes[0].invert_yaxis()

# Non-sarcastic words
words, counts = zip(*non_sarcastic_words)
axes[1].barh(range(len(words)), counts, color='blue', alpha=0.7)
axes[1].set_yticks(range(len(words)))
axes[1].set_yticklabels(words)
axes[1].set_title('Top 20 Words in Non-Sarcastic Comments', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Frequency')
axes[1].invert_yaxis()
```



4.3 Model Implementation

- Logistic Regression and Gradient Boosting:** We built pipelines combining TF-IDF vectorization with the respective classifier. Hyperparameters like regularization strength (C) for Logistic Regression and learning rate and tree depth for Gradient Boosting were tuned using grid search and cross-validation to optimize performance.

```
# Define and train classical ML models
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=500),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Naive Bayes': MultinomialNB()
}
```

- **LSTM and GRU:** Models were built using TensorFlow/Keras with embedding layers initialized randomly and trained on the dataset. Dropout layers were included for regularization. Due to hardware limitations, models were trained on a random subset of the data to manage memory and computation constraints. Early stopping was employed to prevent overfitting.

```
# Build LSTM model
def build_lstm_model():
    model = Sequential([
        Embedding(MAX_VOCAB_SIZE, EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH),
        LSTM(128, dropout=0.5, recurrent_dropout=0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    return model
```

- **RoBERTa:** We used the Hugging Face Transformers library to fine-tune a pretrained RoBERTa model on the sarcasm dataset. This involved adapting the final classification layer and training with an appropriate learning rate schedule. The fine-tuning process benefits from the rich pretraining of RoBERTa, requiring fewer epochs to converge. For usability, a simple Tkinter-based GUI was developed to allow users to input text and receive sarcasm detection scores.

4.4 Model Evaluation

Models were evaluated on a held-out test set using:

- **Accuracy:** Overall proportion of correct predictions.
- **Precision:** Ratio of correctly predicted positive observations to total predicted positives — important for minimizing false positives.
- **Recall:** Ratio of correctly predicted positives to all actual positives — important for minimizing false negatives.

- **F1-Score:** Harmonic mean of precision and recall, providing a balanced measure useful for imbalanced or nuanced classification tasks.

```
def evaluate_model(y_true, y_pred, y_prob, model_name):  
    """  
    Comprehensive model evaluation  
    """  
    from sklearn.metrics import precision_score, recall_score, f1_score  
  
    accuracy = accuracy_score(y_true, y_pred)  
    precision = precision_score(y_true, y_pred)  
    recall = recall_score(y_true, y_pred)  
    f1 = f1_score(y_true, y_pred)  
    roc_auc = roc_auc_score(y_true, y_prob)  
  
    return {  
        'Model': model_name,  
        'Accuracy': accuracy,  
        'Precision': precision,  
        'Recall': recall,  
        'F1-Score': f1,  
        'ROC-AUC': roc_auc  
    }
```

Results:

- **Logistic Regression and Gradient Boosting:** Achieved approximately 71% accuracy and 0.70 F1-score, confirming their effectiveness as strong baselines.
- **LSTM and GRU:** Showed lower accuracy (~61-62%) with signs of overfitting as indicated by fluctuating validation metrics, suggesting the need for further tuning and more data.
- **RoBERTa:** Outperformed other models significantly, leveraging deep contextual representations to capture sarcasm's subtle linguistic cues.

Clean Text:

```
def clean_text(text):  
    """  
    Comprehensive text cleaning function  
    """  
    # Convert to lowercase  
    text = text.lower()  
  
    # Remove URLs  
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)  
  
    # Remove user mentions and hashtags  
    text = re.sub(r'@\w+|#\w+', '', text)  
  
    # Remove HTML tags  
    text = re.sub(r'<.*?>', '', text)  
  
    # Remove extra whitespace  
    text = re.sub(r'\s+', ' ', text).strip()  
  
    return text
```

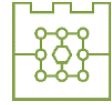
5. Summary

This project successfully applied various natural language processing techniques to the challenging task of sarcasm detection. We implemented and compared multiple models, including traditional supervised learning algorithms (Logistic Regression and Gradient Boosting), recurrent neural networks (LSTM and GRU), and transformer-based architectures (RoBERTa).

Our experiments demonstrated that transformer-based models, particularly RoBERTa, outperform both classical machine learning models and recurrent neural networks in identifying sarcastic expressions. RoBERTa's ability to capture complex contextual dependencies allows it to detect subtle linguistic cues indicative of sarcasm, which often elude simpler models.

While LSTM and GRU models showed promising results, they were limited by longer training times and susceptibility to overfitting. Traditional models such as Logistic Regression and Gradient Boosting provided reliable baselines with moderate accuracy and faster training.

For future work, we recommend expanding the dataset size and diversity, performing extensive hyperparameter tuning on transformer models, and exploring newer architectures such as GPT or LLaMA to further enhance sarcasm detection performance. Additionally, integrating multimodal data and contextual metadata could offer richer insights and improve model robustness.



Through this study, we highlight the potential of advanced NLP models to improve understanding and detection of sarcasm in text, which is essential for enhancing sentiment analysis, social media monitoring, and human-computer interaction systems.

6. References

- [1] Dan Ofer, Sarcasm Detection Dataset, Kaggle, 2018.
- [2] Natekin A., Knoll A. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 2013.
- [3] Logistic Regression - Wikipedia
- [4] Hochreiter S., Schmidhuber J. Long Short-Term Memory. *Neural Computation*, 1997.
- [5] Cho K. et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.
- [6] Liu Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019.