Sarcasm Detection Using Fine-Tuned Transformer Models

Introduction:

Sarcasm detection is a challenging Natural Language Processing (NLP) task that involves NLP tools guessing whether a sentence's intended meaning differs from its literal interpretation. Sarcasm detection asks NLP tools to understand contextual cues, sentiment contrast, and pragmatic knowledge, which is a huge leap from just grasping the meaning of each word in a sentence. Earlier approaches have attempted to overcome the problem by using static word embeddings, such as Word2Vec or GloVe, with classical machine learning classifiers. However, they become outdated with the advent of transformer-based architectures.

The transformer-based architectures use self-attention mechanisms to model contextual dependencies across an entire input sequence. Therefore, it can understand the context of a sentence better to determine whether it's sarcasm. In this project, I'm going to train and evaluate a transformer-based sarcasm detection model and compare its performance against classical baselines. Through quantitative evaluation and error analysis, the project seeks to demonstrate the advantages and limitations of transformer-based models for identifying sarcastic language.

Data Source and Technologies:

The project will use the **Sarcasm Headlines Dataset v2[1]** from Kaggle as the training and testing dataset. The dataset contains news headlines, article links, and an "is_sarcastic" label. It's chosen because it's clean, balanced, and widely used as a benchmark.

In the project, legacy NLP tools for **static word embeddings**, such as Word2Vec and GloVe, will be implemented, and a **neural network** will be trained for the embeddings of Word2Vec and GloVe, respectively. Then, a **transformer-based mode**l from HuggingFace will be finetuned with the sarcasm detection tasks. **NLP techniques for preprocessing** will also be implemented for both legacy tools and the transformer-based model.

Methods:

Three methods for sarcasm detection are implemented and evaluated: Word2Vec with Bidirectional LSTM-GRU, GloVe with Bidirectional LSTM, and BERT Fine-tuning.

Data Preprocessing [2]:
In the project, the data preprocessing step is the same for all three methods. This step first removes the article links. Then it does 5 steps to make the news headlines more formalized:

- HTML Removal: Strip HTML tags using BeautifulSoup

- Bracket Content Removal: Remove text within square brackets
- URL Removal: Remove HTTP/HTTPS URLs
- Stopword Removal: Remove English stopwords and punctuation since they add minimal meaning to the sentence

Word2Vec with Bidirectional LSTM-GRU [2]

- Employ gensim, a Word2Vec model, to pretrain the dataset with a window size of 5 and min_count 1. After the pretraining, each word will be converted to a vector with a dimension of 200.
- Feed the vectors to a neural network for training
- Neural Network Architecture:
    - Embedding layer (trainable, initialized with Word2Vec vectors)
    - Bidirectional LSTM (128 units, 30% dropout, 30% recurrent dropout)
    - Bidirectional GRU (32 units, 10% dropout, 10% recurrent dropout)
    - Dense output layer with sigmoid activation
- Training Configuration:
    - Optimizer: Adam (learning_rate=0.01)
    - Loss: Binary crossentropy
    - Metrics: F1 Score
    - Epoch: 3
    - Batch size: 128
    - Sequence Length: 20
    - Train/Test split: 70/30
- Bidirectional processes context from both directions, with LSTM learns low-level patterns and GRU learns high-level abstractions. The dropout rate will prevent overfitting.

GloVe with Bidirectional LSTM [2]

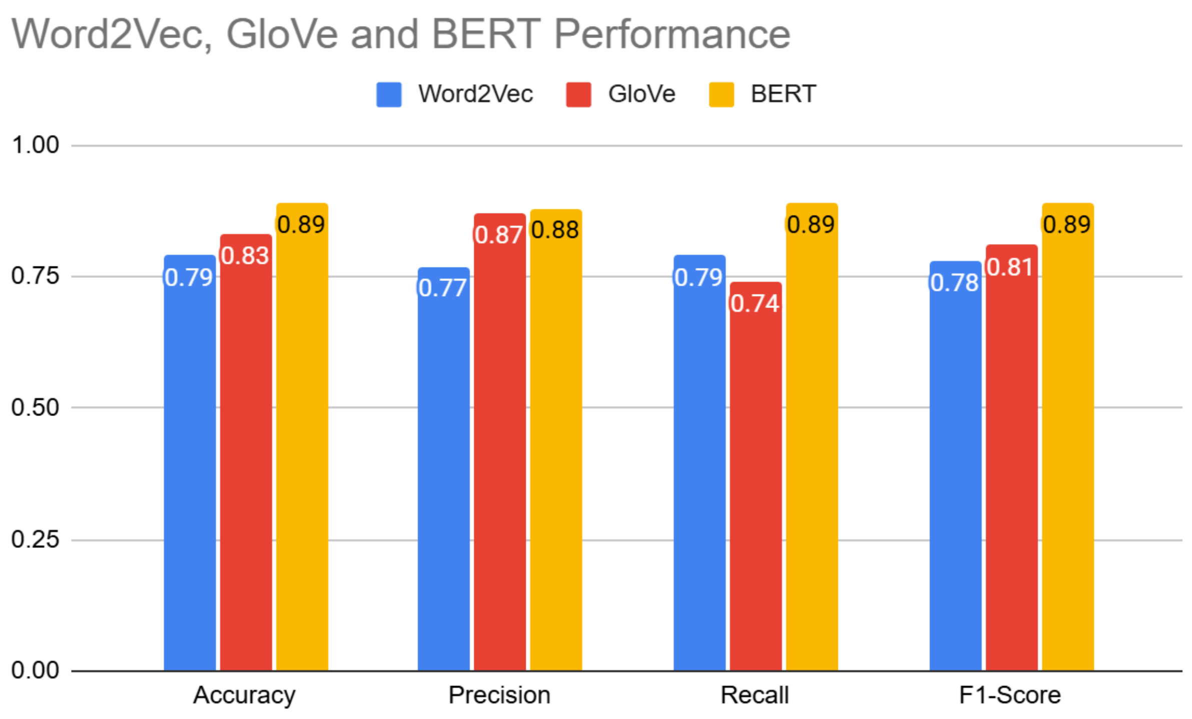
- Use the pre-trained GloVe embeddings (Twitter 27B, 25d) to convert words in the dataset to a vector with dimension 25.
- Feed the vectors to a neural network for training
- Neural Network Architecture:
    - Embedding layer (trainable, initialized with GloVe vectors)
    - Bidirectional LSTM (128 units, 50% dropout, 50% recurrent dropout)
    - Dense output layer with sigmoid activation
- Training Configuration:
    - Optimizer: Adam (learning_rate=0.01)
    - Loss: Binary crossentropy
    - Metrics: F1 Score
    - Epochs: 2

    - Batch size: 128
    - Sequence length: 200
    - Train/Test split: 70/30
- Leverage pre-trained embeddings that capture general Twitter language patterns, and also use longer sequence length (200 to 20) to capture more context while training.

BERT Fine-tuning (Transformer-based LLM)

- Model: BERT-base-uncased (Transformer encoder)
- Architecture: 12 transformer encoder layers, 768 hidden dim, 12 attention heads
- Parameters: ~110 million (fine-tuned for binary classification)
- Tokenization: BERT WordPiece tokenizer with max length 128
- Model Architecture:
    - Pre-trained BERT encoder
    - Classification head (2 output classes)
- Training Configuration:
    - Optimizer: AdamW (learning_rate=2e-5)
    - Loss: Cross-entropy
    - Metrics: Accuracy, Precision, Recall, F1-score
    - Epochs: 5
    - Batch size: 32 (per device)
    - Weight decay: 0.01
    - FP16 mixed precision training
    - Train/Test split: 70/30
    - Evaluation strategy: Per epoch
    - Best model selection: Based on F1-score
- This model gains bidirectional context understanding through transformer self-attention, utilizes state-of-the-art contextual embeddings, and implements FP16 training for BERT to improve speed and memory efficiency.

Results:



The result shows that BERT (Transformer-based LLM) has better performance than both static word embedding methods. Especially, it has a 10% increase in the recall measure, >5% increase in accuracy and F1-score measure, and 1% increase in precision measure. Moreover, the implementation of BERT on the Sarcasm detection task is much more straightforward to implement since feeding embeddings into training and building the training model both have ready-to-use API.

However, since BERT has many more parameters for training, the training time and memory usage dramatically increase compared to the other two methods; for example, it takes 20 times longer to train than the Word2Vec method. Also, the BERT model can overfit with its massive number of parameters.

In the future, the technique LoRA(Low-Rank Adaptation) can be implemented to reduce the memory usage and overfitting effect.

Reference:

1. https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection
2. https://www.kaggle.com/code/madz2000/sarcasm-detection-with-glove-word2vec-83-accuracy