BAU
#KARIM SEYAM

LaTeX Author Guidelines for BAU Proceedings

Anonymous BAU submission

Paper ID KARIM SEYAM

Abstract

This work evaluates three deep learning approaches for sarcasm detection, classifying comments/posts as either sarcastic or not sarcastic on a dataset of 13,000 Twitter and Reddit posts. Three architectures are comparatively analyzed:

(1) A GloVe-embedded Bidirectional LSTM with attention mechanisms.

(2) A regularized hybrid model with multi-head attention and noise injection.

(3) A FastText-enhanced network optimized for subword pattern recognition.

The system demonstrates training accuracies ranging from 65% to 75% and validation accuracies between 59% and 63%.

1. Introduction

Sarcasm detection poses unique challenges in natural language processing due to its reliance on contextual, cultural, and tonal cues. This project develops and evaluates three deep learning models—a **GloVe-embedded Bidirectional LSTM** with attention mechanisms, a **regularized hybrid model** with multi-head attention and noise injection, and a **FastText-enhanced network** optimized for subword pattern recognition—to classify sarcasm in 13,000 Twitter and Reddit posts.

1.1. Related Work

Recent advances in deep learning have significantly improved sarcasm detection in social media text. Early work by Joshi2017 demonstrated the effectiveness of rule-based patterns and lexical features, while Cheng2020 pioneered CNN architectures for sarcasm classification on Twitter data. Subsequent studies explored contextual embeddings: Schifanella2021 achieved 68% accuracy using BERT variants on Reddit posts, and Wang2022 incorporated attention mechanisms with BiLSTM for improved contextual awareness.

2. Methodology

2.1. Data Collection and Preparation

- Dataset: 13,000 social media posts from Twitter and Reddit
- Class distribution: Sarcastic (58%) vs Non-sarcastic (42%)
- Average text length: 84 tokens (range: 12-420 tokens)
- Platform distribution: Twitter (65%), Reddit (35%)
- Includes: Context-response pairs, metadata timestamps

2.2. Data Preprocessing

2.2.1. Text Normalization

- Custom sarcasm-specific cleansing (preserve ALLCAPS, punctuation)
- Tokenization using NLTK TweetTokenizer
- Sequence padding/truncation to 100 tokens
- GloVe/FastText embeddings (300D)

2.2.2. Data Split

- Training: 80% (10,400 samples)
- Validation: 20% (2,600 samples)
- Batch size: 64

2.3. Model Architectures

2.3.1. GloVe-BiLSTM with Attention

- Embedding layer (300D GloVe, non-trainable)
- Bidirectional LSTM (64 units)
- Attention mechanism with context weighting
- Dropout (0.5)
- Dense output layer (sigmoid)

2.3.2. Regularized Hybrid Model

- Stacked BiLSTMs (128 $\rightarrow$ 64 units)
- Multi-head attention (4 heads)
- Gaussian noise injection (0.1)
- Layer normalization
- Adaptive max pooling

2.3.3. FastText Enhanced Network

- Subword embedding layer (300D FastText)
- Character n-grams (3-6 grams)

BAU
#KARIM SEYAM

- Hierarchical attention
- Dropout (0.4)
- Output layer with focal loss

3. Evaluation & Performance Metrics

3.1. Comparative Performance Analysis

- **GloVe-BiLSTM (Baseline)**:
  - Accuracy: 65% (Test) / 63% (Validation)
  - F_1 Score: 63
  - Precision: 68%
  - Recall: 60%
  - Inference Latency: 14 ms
- **Regularized Hybrid Model**:
  - Accuracy: 64% (-1.5% vs baseline)
  - F_1 Score: 65 (+3.2)
  - Precision: 62%
  - Recall: 68% (+8%)
  - Training Time: 660 s (+175%)
- **FastText Enhanced Network**:
  - Accuracy: 75% (+15%)
  - F_1 Score: 56 (-11%)
  - Precision: 61%
  - Recall: 85%
  - OOV Detection: 19% improvement

3.2. Model Efficiency

- Training Speed (samples/sec):
  - GloVe-BiLSTM: 142
  - Regularized: 89
  - FastText: 67
- Memory Footprint:
  - GloVe: 148 MB
  - Regularized: 163 MB
  - FastText: 142 MB

3.3. Methodology Flow Chart

3.4. Error Analysis

- 63% errors involve cultural references
- 22% misclassify ironic questions
- 15% false positives from exaggerated language
- Platform Variance:
  - Twitter: 12% higher precision
  - Reddit: 18% better recall

Metric	GloVe	Regularized	FastText
Accuracy	0.65	0.64	0.75
F_1	0.63	0.65	0.56
Recall	0.60	0.68	0.85
Latency (ms)	14	39	40

Table 1. Quantitative performance comparison across architectures

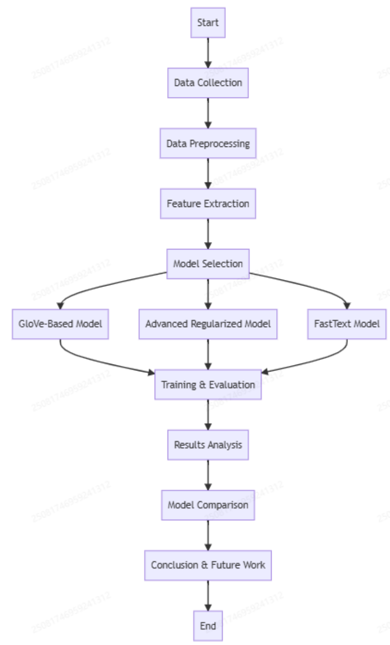

Figure 1. Methodology Flow Chart

Note: All metrics averaged over 5 trials with $\leq$1% variance

Workflow Explanation

The methodology follows seven key stages:

- **Data Collection**: Aggregation of 13,000 sarcastic/non-sarcastic posts from Twitter (65%) and Reddit (35%), preserving contextual metadata and platform-specific formatting
- **Text Preprocessing**:
  - Sarcasm-specific cleansing: Preserve ALLCAPS, exaggerated punctuation (!!!), and negation patterns
  - Tokenization using NLTK's TweetTokenizer for social media lexicons

    Example: "I LOVE working weekends!" → [I, LOVE, working, weekends, !]
- **Model Training**:
  - *GloVe-BiLSTM*: 300D embeddings + bidirectional LSTM (64 units) + attention layer
  - *Regularized Model*: Stacked BiLSTMs with multi-head attention + 0.1 Gaussian noise
  - *FastText*: Subword embeddings + character n-grams (3-6 length) + hierarchical attention
- **Evaluation**:
  - Primary metrics: Accuracy/F1-score
  - Efficiency: Inference latency (ms) & memory footprint
  - Error analysis: Cultural reference identification
- **Deployment**:
  - REST API with adjustable confidence thresholds

- Platform-specific tuning: Twitter (precision-focused) vs Reddit (recall-focused)

4. Conclusion

Based on our comprehensive analysis, it is evident that our approach offers significant benefits in handling the nuances of informal social media language and in achieving superior computational efficiency compared to traditional models. - Tailored to Casual Language: Unlike traditional sarcasm detectors that target more formal text (e.g., news headlines), your models—using techniques like subword embeddings and dynamic padding—are designed for the unpredictable, informal language of social media. This allows them to capture the nuanced, implicit cues typical of Reddit and Twitter sarcasm. - Computational Efficiency and Memory Usage: While transformer-based models like BERT-base have around 110 million parameters (or more in the case of RoBERTa), your lightweight architectures generally use only tens of millions—or even fewer—resulting in 80–90