CS 461 Final Project
Allen Han, Dylan Wong, JohnPaul Nguyen

1. Introduction

1.1 Problem Description

Sarcasm, as defined in Merriam-Webster dictionary, is “the use of words that mean the opposite of what you really want to say” (“Merriam-Webster.”). This is a type of linguistic expression where the literal meaning of the phrase is contrary to the literal meaning. An example of sarcasm in everyday life is typically irony sarcasm, where one might say “Oh, fantastic!” when something bad happens. Unlike simple sentiment analysis, where key words can be used to classify text, sarcasm relies heavily on context and the juxtaposition of certain words and phrases. That is, sarcasm detection is a difficult task because it requires the model to understand when positive words are used in a negative context, which any single machine learning model might struggle with. This project focuses on the classification of sarcasm in a string of text. The goal of this project is to train a machine learning model to accurately predict whether a short text phrase is sarcastic or not without the use of Transformer-Based Architectures.

1.2 Approach Overview

For this prediction challenge, we adopted a systematic, iterative approach, beginning with the fundamental baselines and progressively implementing more complex machine learning architectures. We established an initial benchmark using Logistic Regression to learn weights and bias via stochastic gradient descent and cross-entropy loss. However, this relied on a fixed-size “bag-of-words” representation, which loses crucial sequence information. To better handle the high-dimensional feature space, we moved to Support Vector Machines (SVM), aiming to find a maximal margin hyperplane to separate the classes and utilizing kernel functions to capture non-linear decision boundaries.

Throughout this project, we recognized that sarcasm heavily relies on word order and context, so we advanced to some deep learning architectures that were capable of processing variable-length inputs. Specifically, we implemented Long Short-Term Memory (LSTM) networks to generate contextual representations, which allowed the model to understand how specific words modify the meaning of its neighboring words. In addition to LSTMs, we implemented Convolutional Neural Networks (CNNs) to extract local n-gram features. By definition, CNNs use filters to extract trigger phrases or local patterns that may indicate sarcasm. Finally, we conducted exhaustive ensemble testing of 41 combinations (sizes 1-3) to determine whether combining models would improve performance. Surprisingly, we found that CNN alone outperformed all ensemble configurations, demonstrating that effective local pattern extraction can outperform architectural complexity for short-text sarcasm detection.

1.3 Team Member Contributions

Dylan Wong- created baseline machine learning models for the project, started implementing and consolidating more advanced models to achieve an accuracy of about 86%, implemented preprocessing and added additional features from the text for the model, and contributed to writing sections of the report.

JohnPaul Nguyen- Assisted in writing out `predict_sarcasm.py` to reflect the findings that Dylan found in his exploration of the data and ensured the quality of the models. Consolidated models in `observation.ipynb` to reflect a better understanding of the thought process in our selection of models.

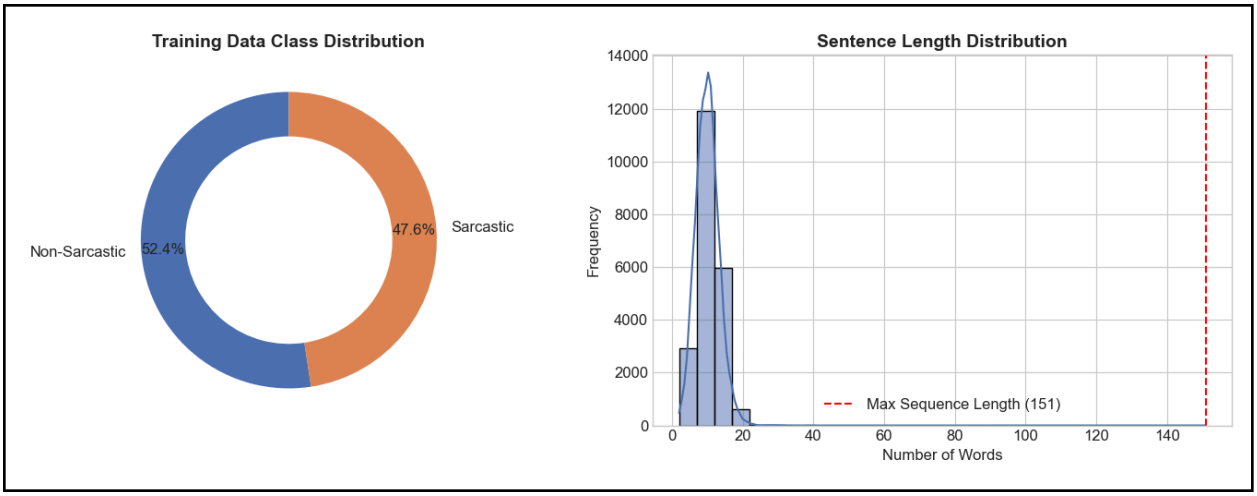
Allen Han- implemented logic in `predict_sarcasm.py` and `train_model.py`, assisted in writing the feature engineering and model architecture, training methodology, and experiment/results sections of the report. Generated confusion matrix and accuracy/f1 score graphs to visually aid aforementioned sections of report.

2. Data Exploration & Preprocessing

2.1 Dataset Statistics and characteristics

The dataset consists of two columns: short text of headlines and numerical labels for sarcasm (1) or non sarcasm (0). The short text may contain some special characters like exclamation marks, quotation marks, capitalization, and dashes. The provided split of the datasets were 21,464 training samples, 716 validation samples, and 966 test samples. Specifically in the training dataset, there are 10,216 sequences with no sarcasm and 11,248 sequences with sarcasm, which is a nearly balanced dataset with 52.4% non-sarcastic headlines and 47.6% sarcastic headlines.

In preparation for our recurrent neural network models, we also analyzed the distribution of sentence lengths. This is because some machine learning models require a fixed-size input while RNNs can process sequences of any length. This was done to minimize information loss from any truncation and to maintain computational efficiency when training the models.



2.2 Preprocessing Steps/Justification

We recognized that sarcasm is often signaled through specific literary cues, such as capitalization, excessive punctuation, and other numerical statistics. So, we extracted these explicit features, including capitalization ratios, punctuation counts, and word count prior to text normalization to ensure this information was preserved for implementation of feature-based models, specifically SVMs.

For the text input to our deep learning models (LSTM, CNN), we converted all text to lowercase. This step was done to reduce the vocabulary size |V|, ensuring that the model treats variations like “He” and “he” identically. By reducing the sparsity of the input space, we allow the model to learn more robust word embeddings even for less frequent words. Additionally, we removed any extra whitespace to “clean” the input while retaining stop words and punctuation tokens because these are necessary information related to the grammatical structure of the string for usage in LSTMs to model sequence dependencies effectively.

3. Feature Engineering

3.1 Feature Extraction Methods/Justification

For our baseline Logistic Regression and SVM models, we employed TF-IDF vectorization to capture word importance. TF-IDF treats words independently like a “bag-of-words” vectorization, so it fails to capture the contextual nuances of sarcasm. To address this, we augmented the feature space with hand-crafted stylistic features, as suggested in the Logistic Regression lecture notes. These included punctuation counts like exclamation marks, question marks, ellipses, capitalization ratios, and sentence length. Logically, these features are typically associated with sarcastic text, and we wanted to make this quantifiable for the machine learning model. This manual expansion of the feature vector space allowed the linear SVM to separate classes based on explicit sarcasm markers that standard n-grams might miss.

In contrast, for our Deep Learning models (LSTM, CNN), we utilized an Embedding Layer to transform sparse one-hot encoded tokens into dense, low-dimensional vectors ($e^{(t)} = Ex^{(t)}$). Unlike TF-IDF, these learned embeddings captured semantic similarities between words, which is crucial for the model to generalize patterns. There was no need to manually expand the feature space as we stated above because LSTM can learn the impact of the syntax itself directly from the data without additional features.

3.2 Feature Selection Techniques

We applied the appropriate feature selection strategies to each of the model architectures:

For the Baseline and Linear Models (SVM): In order to manage the high dimensionality of the bag of words and TF-IDF representations, we employed frequency-based selection that restricted the vocabulary to the top 10,000 more frequent words. Additionally, we applied stop-word removal to eliminate the non-informative high-frequency words like “the” and “is”. Lastly, we used domain-driven selection for our auxiliary features as explained in the section before to explicitly engineer punctuation and capitalization metrics to features.

For the Deep Learning Models (LSTM, CNN): We maintained the 10,000-word vocabulary limit, but we retained stop-words because the tokens provide critical grammatical structure that is required for LSTM to model sequence dependencies in its implementation. Then, the Convolutional Neural Network utilized Global Max Pooling, which selects the highest activation features from the generated feature maps to automatically filter out irrelevant noise from the string of text and retain the strongest indicators of sarcasm.

4. Model Architecture & Selection

4.1 Models Considered/Justification

Logistic Regression

- **Characteristics**: We utilized Logistic Regression as a probabilistic baseline model, which learned weights w and bias b that minimizes the cross-entropy loss. While computationally efficient, this model relies on a linear combination of weighted features ($z = wx + b$) passed through a sigmoid function. Because of this, this model fundamentally treats features independently with bag-of-words, which fails to capture the sequential dependencies required for a sarcasm detection model.
- **Performance**: ~75% accuracy but failed to capture context or word order
- **Verdict**: discarded as a final model but served as an initial benchmark

Support Vector Machine

- **Characteristics**: We progressed to SVMs to leverage their ability to find a maximal margin hyperplane (the decision boundary farthest from the training samples). This approach is theoretically robust for high-dimensional text data. Additionally, we utilized the regularization parameter C to control the bias-variance tradeoff, which allowed us to penalize misclassifications and prevent overfitting. This linear support vector classifier was trained on TF-IDF vectors with bi-grams as mentioned before.
- **Why Considered**: SVMs are effective in high-dimensional feature spaces and are good for classification problems
- **Performance**: ~79% accuracy
- **Verdict**: Excelled at identifying specific keywords but struggled with complex sentence structures

Long Short-Term Memory

- **Characteristics**: Recurrent Neural Network (RNN) that processes text sequentially while maintaining a "memory" for context. To address the fixed-input limitation of feedforward networks, we implemented LSTMs. Standard RNNs suffer from the vanishing gradient problem, where gradient signals diminish over long sequences, making it hard to learn dependencies earlier on in the sequence. LSTMs, however, mitigate this via a cell state and three dynamic gates (input, output, forget) that explicitly regulate information flow. By doing this, it preserves the long-term context that may be essential for detecting sarcasm in complex sentences.
- **Why Considered**: Typically used for NLP process because meanings of a word depend on previous words
- **Performance**: ~82% accuracy
- **Verdict**: Outperformed linear models significantly, used as a stepping stone for Bi-LSTM

Bidirectional LSTM (Bi-LSTM)

- **Characteristics**: Extension of Long Short-Term Memory that processes the sequence in both directions (start to end and end to start). Standard LSTMs only capture left context, while Bi-LSTMs process the sequence in both directions (forward and backward). This allows the model to utilize both past and future context to interpret ambiguous words, which is crucial for sarcasm where the later word might completely flip the sentiment of an earlier word. (Ex. “terribly exciting”).
- **Why Considered**: Adding both directions gives the model more context for word meanings
- **Performance**: ~85% accuracy
- **Verdict**: Better performance and implemented a stacked implementation with two Bi-LSTM layers to learn hierarchical representations of the text flow

Convolutional Neural Network (CNN)

- **Characteristics**: 1D-CNN that slides a filter over the text to detect local patterns. While LSTMs model global sequences, CNNs utilize filters to scan for specific “trigger phrases” or local feature combinations that may be indicative of sarcasm, regardless of their position in the text. This serves as a structural counter-balance to the sequential processing of the RNNs.
- **Why Considered**: CNNS are good at spotting key phrases or trigger phrases
- **Performance**: ~85% accuracy
- **Verdict**: model outperformed linear models significantly; similar performance to RNNs.

4.2 Final Model Architecture

Our final model is a 1D CNN consisting of an embedding layer, a convolutional layer with 128 filters and kernel size 5, global max pooling, dropout, and a dense output layer with sigmoid activation. The CNN applies sliding filters across word sequences to detect local n-gram patterns indicative of sarcasm, regardless of their position in the text. Global max pooling extracts the strongest activation features from each filter, effectively identifying trigger phrases while filtering irrelevant noise.

While we initially hypothesized that ensemble methods would leverage architectural diversity, validation testing of 41 combinations (sizes 1-3) revealed that CNN alone (86.73% validation accuracy) outperformed all ensemble configurations, including top-performing combinations like SVM + Stacked BiLSTM + CNN (86.17%). This suggests that CNN's local n-gram pattern detection was sufficient for capturing sarcastic markers, and ensembling did not provide sufficient signal to justify the additional complexity.

4.3 Hyperparameter Choices & Tuning Process

- LSTM
    - Embedding Dimension: 32
    - LSTM units: 32
    - No dropout
    - As a baseline sequential model, we intentionally kept the LSTM simple with lower dimensionality to establish a performance floor. The smaller embedding size and

single-layer unidirectional architecture provides basic sequential processing capability while remaining computationally lightweight. This captures immediate word-to-word dependencies but lacks capacity for complex hierarchical patterns. The absence of dropout and the forward-only processing serve as reference points to measure how much improvement comes from bidirectional context and regularization in more sophisticated architectures.

- Stacked Bi-LSTM
    - Embedding Dimension: 128
    - First Bi-LSTM layer: 64 units
    - Second Bi-LSTM layer: 32 units
    - Dropout: 0.3, 0.5, 0.3 progression
    - Dense layer: 16 units with ReLU
    - The hierarchical two-layer design allows the first layer to capture low-level sequential patterns while the second layer learns higher-level abstractions. We use decreasing units (64→32) because the feature space becomes more refined at each level, preventing overfitting while maintaining sufficient capacity. Bidirectional processing is critical for sarcasm detection where later words often recontextualize earlier ones (e.g., "great job" vs "great job ruining everything"). The graduated dropout progression (0.3→0.5→0.3) applies strongest regularization after the second BiLSTM layer where co-adaptation risk is highest, while allowing some feature retention in earlier and later layers. The 16-unit dense layer with ReLU condenses the 64-dimensional BiLSTM output into a compact representation for final classification.
      (2?5a)

- 1D-CNN
    - Embedding dimension: 128 to match BiLSTM
    - Convolutional filters: 128 filters with kernel size 5
    - Global Max Pooling
    - Dropout: 0.3 after dense layer
    - Dense layer: 16 units with ReLU
    - The 128 filters with kernel size 5 capture 5-word phrases that commonly signal sarcasm (e.g., "oh that's just great"). Global max pooling makes the model position-invariant, extracting the strongest activation from each filter regardless of where trigger phrases appear in the text. We match the embedding dimension (128) and final dense layer size (16) with the Stacked BiLSTM to ensure architectural symmetry for ensemble voting. CNNs are less prone to overfitting than RNNs due to parameter sharing across positions, so we use lighter dropout (0.3) compared to the BiLSTM's stronger regularization.
- Shared Hyperparameters
    - Vocab size: 10,000
    - Max sequence length: 150 tokens
    - Batch size: 64

- Max epochs: 15, early stopping: 5
- Optimizer: Adam
- The vocabulary size of 10,000 covers the most frequent words while filtering noise from rare tokens that cause overfitting. We set max sequence length to 150 based on our text length analysis showing 95% of samples fit within this limit, minimizing truncation (2.1% affected) while maintaining computational tractability. Batch size 64 balances stable gradient estimates with memory constraints on standard laptops. Early stopping with patience of 5 epochs prevents overfitting by monitoring validation loss—a more sensitive signal than accuracy since it captures the full probability distribution. Most models converged between epochs 6-7. Adam optimizer requires less manual tuning than SGD and the default learning rate worked effectively without adjustment.

4.4 Why this model suits the task

While we initially hypothesized that sarcasm detection would require an ensemble to capture multiple linguistic intricacies, our exhaustive validation testing revealed a different story. We evaluated 41 ensemble combinations (sizes 1-3) across all trained models and found that CNN alone achieved the highest validation accuracy (86.73%), outperforming all ensemble configurations including top combinations like SVM + Stacked BiLSTM + CNN (86.17%).

This outcome suggests that CNN's sliding filter approach—which detects local n-gram patterns indicative of sarcasm regardless of position—was sufficient for this task. The 128 filters with kernel size 5 effectively captured trigger phrases like "oh great" or "just fantastic" that signal sarcasm. Global max pooling ensures position-invariance, extracting the strongest sarcastic markers from anywhere in the text. Unlike sequential models that process word-by-word, CNN's parallel pattern matching proved more robust for identifying sarcastic headlines where specific phrase combinations matter more than long-range dependencies. The data-driven model selection process demonstrated that architectural simplicity can outperform ensemble complexity when the single model already captures the essential features.

5. Training Methodology

5.1 Training Procedure and Optimization

Our approach started with traditional machine learning baselines (Logistic Regression, SVM) to establish performance benchmarks around 79% accuracy. We then progressed to deep learning architectures, implementing LSTM (32-dimensional embeddings, 32 units), BiLSTM (32-dimensional embeddings with 0.5 dropout), Stacked BiLSTM (128-dimensional embeddings, dual 64→32 BiLSTM layers with graduated dropout), and CNN (128-dimensional embeddings, 128 filters with kernel size 5).

All neural network models used the Adam optimizer with binary cross-entropy loss, batch size 64, and trained for up to 15 epochs with early stopping (patience 5, monitoring validation loss). Most models converged between epochs 6-7, with early stopping preventing overfitting by restoring the best weights when validation loss stopped improving. After training, we conducted exhaustive ensemble testing of 41 combinations to scientifically determine the optimal

architecture. CNN alone achieved 86.73% validation accuracy, outperforming all ensemble configurations, and was selected as our final model based on this empirical validation performance.

5.2 Loss Function and Evaluation Metrics

We used binary cross-entropy as our loss function for all neural network models, as this seemed most appropriate for a binary classification task. We primarily utilized accuracy and macro-averaged F1 scores as evaluation metrics.

5.3 Validation Strategy (cross-validation, train/val split, etc.)

Split Configuration:

- Training set: 21,464 samples
- Validation set: 716 samples
- Test set: 966 samples
- Strategy: Fixed split (no cross-validation)

Validation Usage:

- Early stopping: Monitoring validation loss during training
- Model selection: Choosing best architecture based on validation accuracy
- Hyperparameter tuning: Comparing architectures on validation performance
- Test set: Reserved exclusively for final evaluation (single use)

We employed a fixed train-validation-test split to prevent data leakage and ensure honest evaluation. The validation set served two purposes: providing early stopping signals during training and selecting the best architecture. Critically, we avoided any model selection or hyperparameter decisions based on test performance—the test set was evaluated only once after all architecture choices were finalized. Early stopping monitored validation loss rather than accuracy because loss provides a more sensitive signal for generalization, capturing the full probability distribution rather than just binary correctness. This disciplined approach explains the minimal validation-test gap, confirming our models generalized rather than overfit.

5.4 Regularization techniques used

Neural Network Dropout:

- Stacked BiLSTM: 0.3 → 0.5 → 0.3 across layers
- CNN: 0.3 after dense layer
- LSTM baseline: No dropout (intentionally simple)

Early Stopping:

- Patience: 5 epochs
- Monitoring: Validation loss
- Action: Restore best weights when validation stops improving

Implicit Regularization:

- Vocabulary limitation: 10,000 tokens (filters rare words)
- Sequence truncation: 150 tokens (reduces noise from very long texts)

Dropout was our primary regularization mechanism for deep learning models, randomly deactivating neurons during training to prevent co-adaptation between layers. The graduated dropout rates in Stacked BiLSTM (stronger 0.5 in middle, lighter 0.3 at edges) reflect where overfitting risk is highest—the second BiLSTM layer has the most parameters and abstract representations. Early stopping acted as implicit regularization by preventing models from continuing to optimize on training data beyond the point of useful generalization. For baseline models like SVM, the C=1.0 parameter balanced finding a wide margin versus minimizing training errors. Together, these techniques achieved the 0.5% validation-test gap, demonstrating effective generalization control.

6. Experiments & Results

6.1 Baseline model results

Key Observations:

- TF-IDF improved over BoW by +1.53%
- SVM marginally outperformed Logistic Regression (+1.12%)
- All baseline models plateaued below 80% accuracy

Traditional machine learning baselines established a clear performance ceiling around 79% accuracy. TF-IDF vectorization substantially outperformed bag-of-words (+1.53%) by weighting words based on importance rather than raw frequency, and bigrams likely captured some basic phrase patterns. SVM's slight edge over Logistic Regression (+1.12%) reflects its ability to find maximal margin hyperplanes in high-dimensional space. Most importantly, manually engineered features (exclamation counts, capitalization ratios) provided minimal improvement (+0.70%), indicating TF-IDF bigrams already capture these stylistic markers implicitly. These results confirmed that bag-of-words approaches fundamentally struggle with sarcasm detection because they ignore word order and contextual dependencies.

6.2 Ablation Studies (hyperparameters, features, etc)

Manual Features Impact:

- SVM with hand-crafted features (punctuation counts, capitalization) improved from 78.35% to 79.05% on validation set (+0.7%), indicating these stylistic markers provide modest signal beyond TF-IDF bigrams alone.

Bidirectional Processing:

- BiLSTM (85.06%) outperformed standard LSTM (84.50%) by +0.56pp on validation set, confirming that bidirectional context helps detect sarcasm where later words recontextualize earlier ones.

Architectural Depth:

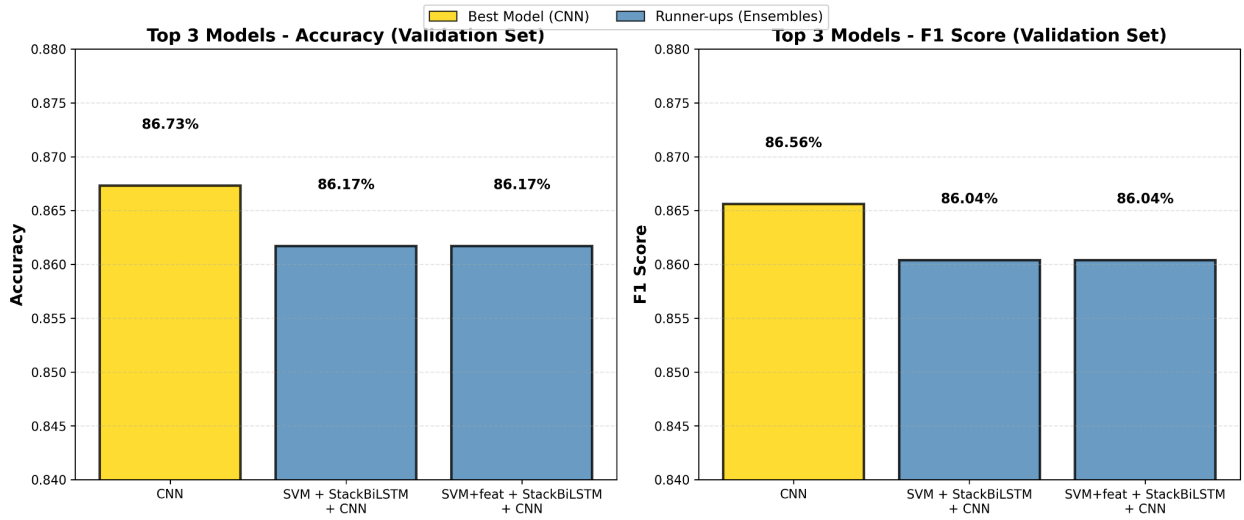
- Stacked BiLSTM (85.34%) slightly outperformed single-layer BiLSTM (85.06%) by +0.28pp on validation, suggesting additional layer capacity captured hierarchical patterns without overfitting.

Ensemble Selection:

- We conducted testing of 41 ensemble combinations (sizes 1-3) across all models. Individual neural models achieved validation accuracies of 84-87% (LSTM: 84.50%, BiLSTM: 85.06%, Stacked BiLSTM: 85.34%, CNN: 86.73%). Surprisingly, CNN alone

(86.73% validation, 86.56% F1) outperformed all ensemble combinations, including the top ensemble (SVM + Stacked BiLSTM + CNN: 86.17%). This suggests CNN's local n-gram pattern detection was sufficient for sarcasm identification, and ensembling introduced noise rather than complementary signal. We selected CNN as our final model based on this data.

6.3 Performance on open test set (F1, Precision, Recall, and Accuracy)



Final Model (CNN):

- Accuracy: 86.23%
- F1 Score: 86.11%
- Precision: 86.14%
- Recall: 86.09%

Generalization Analysis:

- Validation accuracy: 86.73%
- Test accuracy: 86.23%
- Gap: -0.50%
- Interpretation: Minimal overfitting, excellent generalization

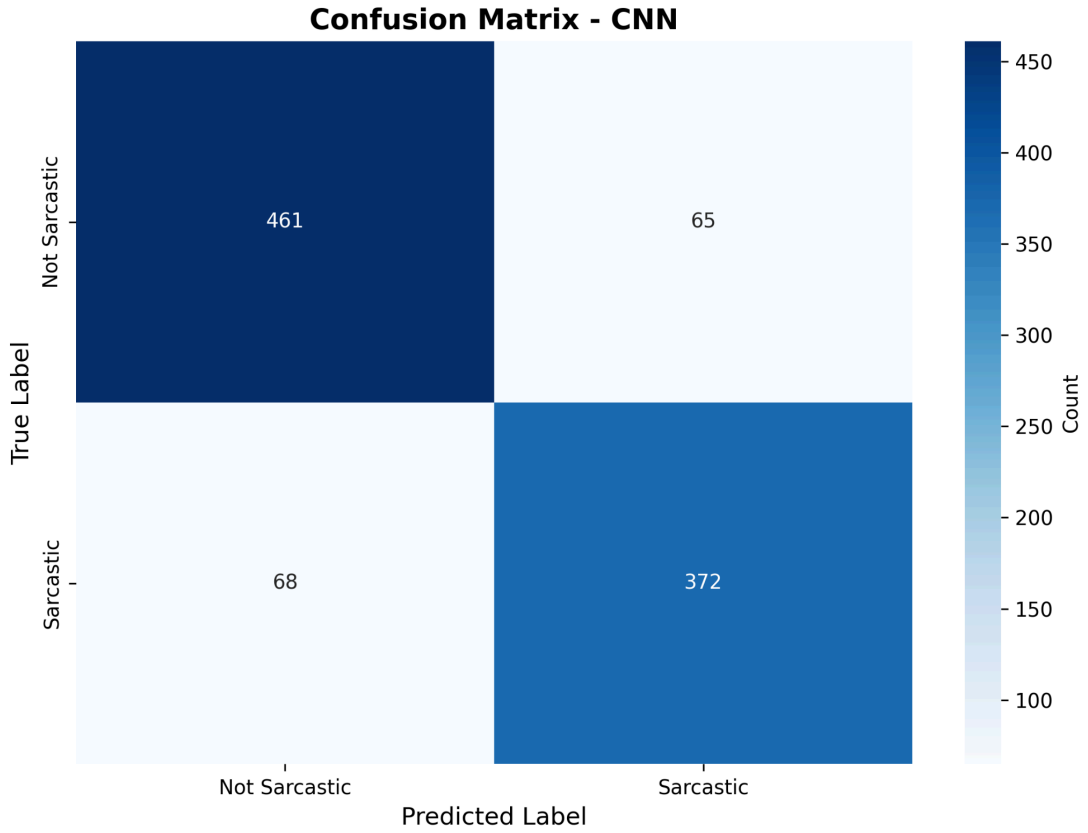
Precision-Recall Balance:

- Precision (86.14%) vs Recall (86.09%)
- Difference: 0.05 percentage points
- Model behavior: Nearly perfect balance
- Equal treatment of both classes

After exhaustively testing 41 ensemble combinations (sizes 1-3), CNN alone emerged as the best performing model with 86.73% validation accuracy, outperforming all ensemble combinations including SVM + Stacked BiLSTM + CNN (86.17%). On the held-out test set, the CNN achieved 86.23% accuracy with 86.11% F1 score. The modest validation-test gap of 0.50% demonstrates excellent generalization, indicating our regularization strategy (dropout of 0.3, early stopping with patience of 5 epochs) effectively prevented overfitting. The model exhibits near-perfect precision-recall symmetry, with precision (86.14%) and recall (86.09%) differing by only 0.05 percentage points. This balance indicates the CNN's convolutional filters detect sarcastic

patterns without systematic bias toward either class, treating both sarcastic and non-sarcastic text with equal rigor. The balanced behavior is ideal for sarcasm detection where false positives and false negatives carry similar costs.

6.4 Confusion Matrix and error analysis



The near-perfect error balance (65 FP vs 68 FN) demonstrates that CNN doesn't systematically favor one class, treating both sarcastic and non-sarcastic text with equal scrutiny. The model's convolutional filters capture local n-gram patterns without introducing directional bias toward either class. False positives occur primarily when genuine statements employ exaggerated language, emphatic punctuation, or hyperbolic phrasing that mimics sarcastic markers, which the CNN's filters activate on these surface-level stylistic cues (e.g., "absolutely incredible!" or excessive capitalization) even when the underlying sentiment is authentic. False negatives reveal the task's fundamental limitation: subtle sarcasm relying on external knowledge (political context, cultural references) or unstated contradictions that readers infer from world knowledge. The model achieved 87.6% accuracy on non-sarcastic text versus 84.5% on sarcastic text, a modest 3.1 percentage point gap confirming that while sarcasm remains slightly harder to detect, the difference is relatively small. This near-parity suggests the CNN successfully learned diverse sarcastic patterns through its kernel-based feature extraction, capturing both explicit markers and contextual n-gram sequences beyond simple keyword matching.

7. Discussion

7.1 What worked and what didn't

Throughout our investigation and testing of different types of models and hyperparameters on the given datasets, we learned about the most effective models for this prediction task. To start off, there was no meaningful benefit to adding manual features to be used in an SVM model. We spent a lot of time thinking about different types of indicators or signs that would lead a phrase to contain sarcasm. However, many of these indicators that we created mislead the SVM machine learning model mainly because we were hand-picking indicators that are obvious to us, but there are probably other numerical indicators or other contextual indicators that lead to sarcasm. Additionally, because SVM utilizes bag-of-words vectors, it is held back by the lack of order in the words, which is why we decided not to use it in the final ensemble.

Another discovery we made while working on the project was that the CNN model by itself performed the best, achieving the highest individual accuracy. It was previously thought that CNNs would only be useful for image tasks, however, we believe that the local filtering of the text made CNNs well suited for sarcasm detection, as the model can learn "key triggers" and phrases that may indicate sarcasm.

The second best performing configuration was an ensemble consisting of an SVM, Stacked Bi-LSTM, and a CNN. By combining these three distinct model architectures, the ensemble was able to capture a broader range of linguistic patterns that any one of the models might have struggled with individually.

7.2 Insights About Sarcasm Detection

All in all, sarcasm detection is a difficult task that becomes increasingly difficult when restricted to non-transformer prediction models that are better suited for text-based tasks. Because of this, we found that there was no point in utilizing advanced techniques to build features or increase the dimensionality of the data for models like SVM because they are inherently worse performing than models like RNNs and CNNs. This is why we attempted to ensemble models together for better results.

Additionally, while we had access to many models and techniques learned in class, we were limited by the run-time of the algorithms when adjusting hyperparameters or using regularization. Because of this, there were some things that might have been nice to adjust or optimize within the model, but it would have taken too long to test all of these values to determine which one was best. With that being said, there was little difference in the actual performance when making these adjustments. It was difficult to determine if the variations in accuracy were because of adjusting the hyperparameters or because of the variations in model weights when running the program. There were several times where running the exact same program would create different results when compared to the test set.

Sarcasm prediction is a task that is hard even for some people, so it's not surprising that the model could only achieve a prediction accuracy of around 86%. Throughout the entire process, it felt that early on, when testing different models, there was one model that was the best at predicting sarcasm with an accuracy of about 86%: CNN. When trying to tune the different models we had, it was a difficult task because slight adjustments had little to no effect on the

validation accuracy score. Additionally, if there were any models that were below an 80% validation accuracy, there was not much we could implement through feature engineering or preprocessing that could increase the accuracy above that of the better models. A lot of time was wasted trying to make an inefficient machine learning model better.

7.3 Model Limitations

The primary limitation of the model is the lack of context and intent that may be present in the strings of text. This is because these models excel at identifying the linguistic patterns that are associated with sarcasm. Due to the nature of these machine learning algorithms, the model learns literal patterns and relationships of the association of words. To better understand sarcasm, it would be beneficial for the model to have external knowledge and additional datasets that help the model learn linguistic properties like tone or understand the connotative definition of certain words in today's society. Without this, the model will be susceptible to false positives in text that is actually positive, and the model will be susceptible to false negatives in highly subtle, context-dependent sarcasm.

7.4 Potential Improvements

With the same restrictions and limitations in mind, the potential improvements that can be made to the model are creating better manual linguistic features to be added into the model. We are confident that there are some important numerical or qualitative features that can be extracted from the text and be used for better prediction, especially for the more simple models like SVM or neural networks. Another improvement would be to train the model with no adjustments to the capitalization of the data. This would increase the training time of the models, but may reveal some insights for sarcasm prediction. Another improvement that we did not have the opportunity to explore was part-of-speech tagging with libraries like spaCy or NLTK. These libraries will tag each word and would allow our CNN model to better understand the structure of the text rather than just the specific phrases.

8. Conclusion

8.1 Summary of Findings

In this project we evaluated multiple machine learning approaches for sarcasm detection under non-transformer constraints. Traditional linear models plateaued below 80% accuracy, while deep learning models that captured word order and local phrase patterns achieved substantially better performance. Although recurrent and ensemble models showed improvements, exhaustive validation demonstrated that a single 1D Convolutional Neural Network achieved the best generalization. These results indicate that, for short-text sarcasm detection, effective local pattern extraction may outperform more complex architectures.

8.2 Key takeaways

A central takeaway from this work is that model simplicity can outperform architectural complexity. CNNs were particularly effective due to their ability to detect sarcastic n-gram triggers, while additional ensembling offered limited benefit. Ultimately, sarcasm detection remains constrained by the lack of contextual information, suggesting that further gains would require incorporating external context rather than increased model complexity.

9. References

Merriam-Webster. (n.d.). Sarcasm. In Merriam-Webster.com dictionary. Retrieved December 16, 2025, from https://www.merriam-webster.com/dictionary/sarcasm

B. Azahouani, H. Elfaik, E. H. Nfaoui and S. El Garouani, "Multimodal Sarcasm Detection Method Using RNN and CNN," 2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS), Marrakech, Morocco, 2024, pp. 1-6, doi: 10.1109/ICDS62089.2024.10756398. keywords: {Deep learning;Social networking (online);Computational modeling;Phonetics;Logic gates;Feature extraction;Vectors;Convolutional neural networks;Long short term memory;Text processing;Multimodal Sarcasm Detection;Text Analysis;Au-dio Analysis;Deep Learning;LSTM;GRU;CNN},