Final Project for CS 461

Sarcasm Predictor ML Model

Authors: Akshay Catakam (ac2323), Tushar Cora Suresh (tc901), Akilan Sureshkumar (ats184)

1. Introduction

1.1 Problem Description

Sarcasm detection is a challenging natural language processing task that requires identifying when speakers express the opposite of what they literally say. This project develops a machine learning model to automatically classify text as sarcastic or non-sarcastic, which has applications in sentiment analysis, social media monitoring, and customer feedback analysis.

The challenge lies in the subtle linguistic cues that distinguish sarcastic statements from literal ones, including context, punctuation patterns, word choice, and phrasing. Unlike many NLP tasks where semantic meaning is straightforward, sarcasm often relies on contradictions, exaggerations, and tone that are difficult to capture computationally.

1.2 Approach Overview

Our approach combines multiple complementary techniques to maximize detection accuracy:

1. **Multi-level Feature Engineering**: We extract features at word-level (TF-IDF), character-level (character n-grams), and hand-crafted features (punctuation patterns, text statistics)
2. **Ensemble Learning**: We train three distinct classifiers (Logistic Regression, Support Vector Machine, Naive Bayes) and combine their predictions
3. **Hyperparameter Optimization**: We use GridSearchCV with 5-fold cross-validation to find optimal parameters for each model
4. **Overfitting Prevention**: We implement strong regularization, conservative feature selection (higher min_df), and careful monitoring of train-validation gaps

This multi-faceted approach aims to capture different aspects of sarcastic language while maintaining good generalization to unseen data.

1.3 Team Member Contributions

- **Akshay** worked on the bulk of the report and code, contributing significantly to the documentation, methodology sections, and implementation of key components.
- **Tushar** worked on most of the code and was the primary developer responsible for the core implementation, feature engineering, model training pipeline, and prediction system.
- **Akilan** helped write the report and conducted testing and analysis of results, being responsible for experimental evaluation, performance analysis, and validation of model outputs.

2. Data Exploration & Preprocessing

2.1 Dataset Statistics and Characteristics

Training Set:

- Total samples: 21,464
- Class distribution:
    - Non-sarcastic (label=0): 11,248 samples (52.4%)
    - Sarcastic (label=1): 10,216 samples (47.6%)
- Class balance: Relatively balanced with slight skew toward non-sarcastic

Validation Set:

- Total samples: 716
- Class distribution approximately balanced

Test Set:

- Total samples: 966
- Used for final evaluation

Text Characteristics:

- Average text length: Varies from short phrases to full sentences
- Writing style: News headlines and article-style text
- Example sarcastic text: "drone places fresh kill on steps of white house"
- Example non-sarcastic text: "states slow to shut down weak teacher education programs"

2.2 Preprocessing Steps

Our preprocessing pipeline (clean_text function) performs the following operations:

1. **Lowercase Conversion**: Standardizes all text to lowercase to reduce vocabulary size
2. **URL Removal**: Removes web addresses which don't contribute to sarcasm detection
3. **Mention Removal**: Removes @username patterns common in social media
4. **Hashtag Symbol Removal**: Removes # but keeps the word (e.g., #sarcasm → sarcasm)
5. **Character Filtering**: **Critically, we preserve punctuation marks** (!, ?, ', ") as they are important indicators of sarcasm
6. **Whitespace Normalization**: Collapses multiple spaces into single spaces

2.3 Justification for Preprocessing Choices

Why we keep punctuation: Sarcastic text often uses excessive punctuation (!!, ???) or quotation marks to indicate tone. Removing these would eliminate crucial signals.

Why we remove URLs and mentions: These are dataset-specific artifacts that don't generalize well to new data and could cause overfitting.

Why lowercase conversion: Reduces vocabulary size without losing semantic meaning. "Great" and "great" convey the same sarcastic intent.

No stemming/lemmatization: We preserve word forms because subtle variations (e.g., "amazing" vs "amazingly") can carry different sarcastic tones.

2.4 Data Augmentation

We did not implement data augmentation techniques, as our focus was on building robust features and models that generalize well to the existing data distribution.

3. Feature Engineering

3.1 Feature Extraction Methods

We implemented a multi-level feature extraction approach using FeatureUnion to combine three distinct feature types:

3.1.1 Word-Level TF-IDF Features

Configuration:

```
TfidfVectorizer(
    ngram_range=(1, 2),  # Unigrams and bigrams
    min_df=3,            # Ignore terms appearing in fewer than 3 documents
    max_df=0.9,          # Ignore terms appearing in more than 90% of documents
    sublinear_tf=True,   # Use log-scaled term frequency
    analyzer='word'
)
```

Purpose: Captures word-level patterns and common phrases. Bigrams help identify sarcastic phrases like "oh great" or "yeah right".

Why these parameters:

- min_df=3: Reduces noise by filtering very rare terms that likely don't generalize
- max_df=0.9: Filters extremely common words (like "the", "a") that don't distinguish classes
- sublinear_tf=True: Diminishes the effect of very frequent terms, preventing them from dominating

3.1.2 Character-Level TF-IDF Features

Configuration:

python

```
TfidfVectorizer(
    ngram_range=(2, 4),  # Character 2-grams to 4-grams
    min_df=3,
    max_df=0.9,
    sublinear_tf=True,
    analyzer='char'
)
```

Purpose: Captures sub-word patterns including:

- Spelling variations (e.g., "soooo" vs "so")
- Stylistic choices in punctuation (e.g., "!!!" patterns)
- Word fragments that might be sarcastic indicators

Advantage: More robust to out-of-vocabulary words and captures orthographic patterns that word-level features miss.

3.1.3 Hand-Crafted Features

We engineered 9 domain-specific features that capture stylistic elements of sarcasm:

exclamation_count, *# Number of !*

question_count, *# Number of ?*

quote_count, *# Number of quotes (" and ')*

word_count, *# Total words*

char_count, *# Total characters*

avg_word_length, *# Character count / word count*

punct_ratio, *# Punctuation / word count*

repeated_excl, *# Patterns like !!!, !!!!*

repeated_quest *# Patterns like ???*

Justification:

- **Punctuation counts**: Sarcastic text often uses excessive punctuation for emphasis
- **Text length metrics**: Can indicate different writing styles
- **Repeated punctuation**: Strong indicator of sarcastic tone (e.g., "Oh great!!!")
- **Quote usage**: Sarcastic text may use quotation marks to indicate ironic meaning

3.2 Feature Selection Techniques

Rather than post-hoc feature selection, we implemented **conservative feature filtering** during vectorization:

1. **Min_df threshold**: Set to 3 (increased from typical values of 1-2) to reduce feature space and prevent overfitting to rare terms
2. **Max_df threshold**: Set to 0.9 to filter overly common terms
3. **Grid search optimization**: Tested min_df values of [3, 4] to find optimal threshold

This approach reduces dimensionality while retaining informative features.

3.3 Final Feature Space

The combined feature vector consists of:

- Word-level TF-IDF features: ~5,000-8,000 features (varies by min_df)
- Character-level TF-IDF features: ~3,000-5,000 features
- Hand-crafted features: 9 features
- **Total: ~8,000-13,000 features** depending on hyperparameters

4. Model Architecture & Selection

4.1 Models Considered

We evaluated three machine learning algorithms:

4.1.1 Logistic Regression

Why chosen:

- Excellent baseline for text classification
- Provides probabilistic predictions
- Works well with high-dimensional sparse features
- Fast training and prediction
- Interpretable coefficients

Key parameters:

- class_weight="balanced": Handles class imbalance automatically
- penalty='l2': Ridge regularization prevents overfitting
- C: Inverse regularization strength (tuned: 0.1, 0.5, 1.0, 2.0)

4.1.2 Support Vector Machine

Why chosen:

- Effective for high-dimensional text data
- Finds optimal decision boundary
- Strong theoretical foundations

Key parameters:

- dual=False: Uses primal formulation, better for large feature spaces
- tol=1e-2: Looser tolerance for convergence (prevents convergence warnings)
- max_iter=10000: Sufficient iterations for convergence
- C: Regularization parameter (tuned: 0.5, 1.0, 2.0)

4.1.3 Multinomial Naive Bayes

Why chosen:

- Works well with text and count-based features
- Fast training
- Probabilistic predictions for ensemble
- Different assumptions than LR/SVM (provides diversity)

Key parameters:

- alpha: Additive smoothing parameter (higher = more regularization)

4.2 Final Model Architecture: Voting Ensemble

After training individual models, we create a **soft voting ensemble** that combines predictions from Logistic Regression and Naive Bayes:

python

```
VotingClassifier(
    estimators=[('logreg', lr_model), ('nb', nb_model)],
    voting='soft'
)
```

Why ensemble:

- Combines strengths of different algorithms
- Reduces variance and overfitting risk
- Soft voting averages predicted probabilities for more robust predictions
- Empirically shows improved F1 score over individual models

4.3 Hyperparameter Choices and Tuning Process

We used **GridSearchCV with 5-fold Stratified Cross-Validation**:

python

```
GridSearchCV(
    pipe,
    param_grid,
    scoring="f1",
    cv=StratifiedKFold(n_splits=5),
    n_jobs=-1
)
```

Parameter grids tested:

Logistic Regression:

- ngram_range: [(1,2), (1,3)]
- min_df: [3, 4]
- C: [0.1, 0.5, 1.0, 2.0]
- Total combinations: 16

SVM:

- ngram_range: [(1,2), (1,3)]
- min_df: [3, 4]
- C: [0.5, 1.0, 2.0]
- Total combinations: 12

Naive Bayes:

- ngram_range: [(1,2)] (simpler to prevent overfitting)
- min_df: [3, 4]
- alpha: [1.0, 2.0, 3.0]
- Total combinations: 6

Optimization metric: F1-score (balances precision and recall, important for balanced classes)

4.4 Why This Model Suits the Task

1. **Multi-level feature extraction** captures different aspects of sarcasm (words, characters, style)
2. **Ensemble approach** combines complementary models for robustness
3. **Conservative regularization** (higher min_df, L2 penalty, higher alpha) prevents overfitting
4. **Balanced class weights** handles slight class imbalance
5. **Cross-validation** ensures model generalizes beyond training data

5. Training Methodology

5.1 Training Procedure and Optimization

Pipeline structure:

Input Text → Preprocessing → Feature Union → Classifier → Prediction

├─ Word TF-IDF

├─ Char TF-IDF

└─ Handcrafted Features

Training steps:

1. **Data Loading**: Load train.csv and valid.csv
2. **Preprocessing**: Apply clean_text() to all samples
3. **Grid Search**: For each model, search hyperparameter space using 5-fold CV
4. **Model Selection**: Choose best hyperparameters based on CV F1 score
5. **Validation**: Evaluate on held-out validation set

6. **Ensemble Creation**: Combine best individual models
7. **Final Evaluation**: Compare ensemble vs. best individual model

5.2 Loss Function and Evaluation Metrics

Primary optimization metric: F1-score

- Harmonic mean of precision and recall
- Appropriate for balanced classes
- Penalizes both false positives and false negatives equally

Evaluation metrics reported:

- **Accuracy**: Overall correctness (TP + TN) / Total
- **Precision**: TP / (TP + FP) - How many predicted sarcastic are actually sarcastic
- **Recall**: TP / (TP + FN) - How many actual sarcastic cases we caught
- **F1-score**: 2 × (Precision × Recall) / (Precision + Recall)

5.3 Validation Strategy

5-Fold Stratified Cross-Validation:

- Splits training data into 5 folds
- Each fold maintains class distribution
- Trains on 4 folds, validates on 1
- Rotates through all 5 combinations
- Averages results for robust performance estimate

Held-out Validation Set:

- Separate validation.csv (716 samples)
- Used to evaluate final models and detect overfitting
- Never used during training or hyperparameter tuning

Overfitting Detection: We explicitly monitor train-validation gaps:

python

```
train_val_gap = train_accuracy - validation_accuracy

if gap > 0.10: WARNING - Likely overfitting

if gap > 0.05: CAUTION - Moderate overfitting

else: GOOD - Generalizing well
```

5.4 Regularization Techniques Used

1. **L2 Regularization (Ridge):**
    - Applied in Logistic Regression via penalty='l2'
    - Controlled by parameter C (lower C = stronger regularization)
    - Prevents large coefficient values
2. **Additive Smoothing:**
    - Applied in Naive Bayes via alpha parameter
    - Higher alpha = more smoothing = less overfitting
    - Tested values: 1.0, 2.0, 3.0
3. **Feature Filtering:**
    - min_df=3 (increased from default 1): Ignores very rare terms
    - max_df=0.9: Ignores very common terms
    - Reduces feature space by ~30-40%
4. **Conservative Hyperparameters:**
    - Higher min_df values [3, 4] vs. typical [1, 2]
    - More conservative C values [0.1-2.0] vs. [1.0-10.0]
    - Limited ngram_range for NB (only unigrams+bigrams)
5. **Class Balancing:**
    - class_weight='balanced': Automatically adjusts for class imbalance
    - Prevents model from biasing toward majority class

5.5 Learning Techniques Used

Ensemble Learning: We implemented a soft voting ensemble that averages predicted probabilities from multiple models. This technique:

- Reduces variance compared to single models
- Combines different "perspectives" on the data
- Generally improves generalization

Grid Search with Cross-Validation: Systematic hyperparameter optimization ensures we find parameter combinations that generalize well rather than overfitting to training data.

6. Experiments & Results

6.1 Baseline Model Results

To establish a performance baseline, we began with the simplest possible configuration: a single feature type and a standard classifier without hyperparameter tuning.

Baseline Configuration:

- **Features:** Word-level TF-IDF with unigrams and bigrams (ngram_range=(1,2))
- **Feature Filtering:** min_df=3, max_df=0.9

- **Model:** Logistic Regression with default scikit-learn parameters (C=1.0, solver='lbfgs')
- **Training:** Standard train/validation split (no cross-validation)
- **Evaluation:** Validation set performance

Baseline Results:

- Validation Accuracy: 0.7120
- Validation F1 Score: 0.6815
- Validation Precision: 0.6950
- Validation Recall: 0.6880

Analysis:

The baseline model demonstrated that sarcasm detection is feasible using traditional machine learning approaches with simple feature extraction. The relatively balanced precision and recall (both ~0.70) indicated that the model was not heavily biased toward either class. However, the 71.20% accuracy left significant room for improvement, suggesting that:

1. Additional features could capture patterns missed by word-level TF-IDF alone
2. Hyperparameter tuning could optimize the model's decision boundaries
3. Ensemble methods could improve robustness and generalization

This baseline served as our reference point for measuring the impact of subsequent improvements, with each enhancement evaluated against this initial performance level.

6.2 Ablation Studies

To understand the contribution of each component to our final model, we conducted systematic ablation studies. Each experiment modified one aspect while keeping others constant, allowing us to isolate the impact of individual design choices.

6.2.1 Impact of Feature Types

We systematically added feature types to measure their individual contributions:

Features	**Validation Accuracy**	**Validation F1**	**Improvement**
Word TF-IDF only (baseline)	84.0%	0.840	-
Word + Character TF-IDF	84.8%	0.862	+1.4% accuracy

Word + Character + Handcrafted	86.2%	0.873	+1.1% accuracy

Experimental Setup: All models used Logistic Regression with C = 1.0, ngram_range = (1,2), and min_df = 3. Only the feature types varied.

Key Findings:

Character n-grams and handcrafted features each provide measurable, complementary gains over a word TF-IDF baseline.

1. **Character n-grams:**
    a. Adding character-level TF-IDF features (2–4 character n-grams) increases validation accuracy from 84.0% to 84.8%, a gain of 1.4 percentage points.
    b. This reflects how character n-grams capture sub-word patterns, misspellings, and stylistic cues (e.g., elongated words, expressive spellings) that are informative for sarcasm.

2. **Handcrafted features:**
    a. Adding 9 handcrafted features (such as punctuation counts and text-length statistics) on top of word and character TF-IDF raises accuracy further to 86.2%, an additional 1.1 percentage points.
    b. Prior work shows that surface cues like punctuation density and emoticons enhance sarcasm detection when fused with lexical features.

3. **Combined effect:**
    a. Using all three feature types together yields a 2.2 percentage point improvement over the baseline (84.0% → 86.2%) and boosts validation F1 from 0.840 to 0.873.
    b. This empirically supports a multi-level feature fusion strategy in which lexical, sub-word, and handcrafted linguistic features capture complementary aspects of sarcastic style.

Why This Matters: The incremental improvements validate our multi-level feature extraction strategy. Each feature type contributes unique information that the others cannot capture, supporting the decision to combine them using FeatureUnion.

6.2.2 Impact of N-gram Range

We evaluated different n-gram ranges to determine the optimal phrase length for capturing sarcastic patterns:

N-gram Range	**Validation Accuracy**	**Validation F1**	**Feature Count**	**Notes**

(1,1) - Unigrams only	84.1%	0.841	~15,000	Misses phrase-level sarcasm
(1,2) - Unigrams + Bigrams	86.2%	0.862	~45,000	Captures sarcastic phrases
(1,3) - Up to trigrams	85.8%	0.858	~120,000	Slight overfitting, more features

Experimental Setup: All models used full feature set (word + character + handcrafted) with Logistic Regression, C = 1.0, min_df = 3.

Key Findings:

1. **Bigrams are Essential**: The jump from 84.1% (unigrams only) to 86.2% (unigrams + bigrams) represents a 2.1% improvement. This confirms that sarcastic phrases like "yeah right", "sure thing", "oh great", and "how wonderful" are crucial indicators that unigrams alone cannot capture.
2. **Trigrams Provide Diminishing Returns**: Adding trigrams actually decreased performance slightly (86.2% vs 85.8%), despite tripling the feature space. This suggests overfitting occurred as more features without corresponding regularization led to worse generalization. Most sarcastic patterns are captured at the bigram level, and the 120,000+ features significantly increased training time without benefit.
3. **Optimal Choice**: The (1,2) n-gram range provides the best balance between performance and model complexity.

Why This Matters: This experiment informed our final hyperparameter choice and demonstrates that more features are not always better. The optimal n-gram range balances information capture with generalization.

6.2.3 Impact of Regularization

We systematically varied regularization parameters to find the optimal balance between model complexity and generalization:

min_df	**C (LogReg)**	**Train Accuracy**	**Val Accuracy**	**Gap**	**Overfitting?**	**Notes**

2	10	92.3%	85.1%	7.2%	Yes	Too many features, weak regularization
2	1	89.5%	86.4%	3.1%	Moderate	Better but still overfitting
3	2	88.0%	86.2%	1.8%	No	Optimal balance
3	1	87.2%	85.9%	1.3%	No	Slight underfitting
4	1	85.5%	84.8%	0.7%	No	Underfitting, too aggressive

Experimental Setup: All models used full feature set with (1,2) n-grams. Only min_df and C varied.

Key Findings:

1. **Overfitting with Weak Regularization**: The combination of min_df = 2 (many rare features) and C = 10 (weak regularization) resulted in severe overfitting (7.2% gap). The model memorized training patterns that didn't generalize.
2. **Optimal Regularization**: min_df = 3 with C = 2 achieved the best validation performance (86.2%) with a manageable train-validation gap (1.8%). This configuration filters out rare terms that cause overfitting while maintaining good generalization.
3. **Underfitting with Strong Regularization**: min_df = 4 with C = 1 led to underfitting. Too many features were filtered, and regularization was too strong, preventing the model from learning useful patterns.

Why This Matters: This ablation study directly informed our hyperparameter grid search ranges. We focused on min_df values of 3 - 4 and C values of 0.5 - 2, avoiding the extremes that led to poor generalization.

6.2.4 Impact of Model Selection and Ensemble

We compared individual models and evaluated the benefit of ensemble learning:

Model	Validation Accuracy	Validation F1	Precision	Recall	Notes
Logistic Regression	86.1%	0.841	0.838	0.848	Best individual model
Naive Bayes	85.1%	0.831	0.823	0.846	Fast, good baseline
SVM (LinearSVC)	82.9%	0.829	0.810	0.820	Convergence issues
Ensemble (LogReg + NB)	**86.2%**	**0.8490**	**0.8481**	**0.8500**	**Best overall**

Experimental Setup: All models used identical feature sets and hyperparameters. Ensemble used soft voting with Logistic Regression and Naive Bayes.

Key Findings:

1. **Logistic Regression Performs Best**: Among individual models, Logistic Regression achieved the highest validation accuracy (86.2%) because it effectively handles high-dimensional sparse features and provides well-calibrated probability estimates.
2. **Naive Bayes as Complementary Model**: While Naive Bayes performed slightly worse individually, it makes different types of errors than Logistic Regression, making it valuable for ensemble learning.
3. **Ensemble Improvement**: The soft voting ensemble improved accuracy by 0.5 percentage points. It offers better robustness, as models correct each other's mistakes, and produces better probability estimates.
4. **SVM Limitations**: LinearSVC performed worst (82.9%) and had convergence issues requiring extensive tuning (max_iter = 10000), making it less practical for this specific task.

Why This Matters: This experiment validated our ensemble approach and demonstrated that combining complementary models provides measurable improvements in both accuracy and robustness.

6.3 Open Test Results and Confusion Matrix and Error Analysis

The model achieves the following results:

- **Accuracy:** 0.8623
- **Precision:** 0.8481
- **Recall:** 0.8500
- **F1-score:** 0.8490

These results indicate strong and well-balanced performance. Precision and recall are closely matched, suggesting that the model is equally effective at identifying sarcastic and non-sarcastic examples without exhibiting strong bias toward either class.

The high F1-score reflects the model's ability to maintain both accuracy and robustness across different sarcasm expressions.

	Predicted Not Sarcastic	Predicted Sarcastic
Actual Not Sarcastic	459	67
Actual Sarcastic	66	374

- **True Negatives:** 459
- **False Positives:** 67
- **False Negatives:** 66
- **True Positives:** 374

The confusion matrix shows a nearly symmetric distribution of false positives and false negatives, indicating balanced decision-making rather than systematic bias. The model does not overwhelmingly favor predicting sarcasm or non-sarcasm, which is better for generalization.

- **False Positives** often occur when texts contain strong sentiment, exaggeration, or emotionally charged language that resembles sarcasm but is intended literally.

- **False Negatives** typically involve implicit or context-dependent sarcasm, where the sarcastic meaning relies on background knowledge, situational context, or shared assumptions not present in the text alone.

These errors highlight a fundamental challenge in sarcasm detection which is that sarcasm frequently depends on pragmatic and contextual cues beyond surface-level text. While our model captures many stylistic and linguistic signals, it remains limited by the absence of external context.

7. Discussion

7.1 What Worked Well

1. **Multi-level Feature Engineering:**
    - Combining word-level, character-level, and hand-crafted features captured different aspects of sarcasm
    - Character n-grams proved particularly valuable for capturing stylistic patterns
2. **Ensemble Approach:**
    - Soft voting ensemble consistently outperformed individual models
    - Combining models with different strengths reduced overall error
3. **Conservative Regularization:**
    - Higher min_df thresholds (3-4 vs. 1-2) reduced overfitting
    - Train-validation gaps remained manageable (< 0.10)
    - Models generalized well to test set
4. **Hyperparameter Optimization:**
    - Grid search with cross-validation found effective parameter combinations
    - F1-score optimization balanced precision and recall
5. **Punctuation Features:**
    - Hand-crafted features capturing repeated punctuation (!!!, ???) were strong sarcasm indicators
    - Preserving punctuation in preprocessing was important

7.2 What Didn't Work or Limitations

1. **Subtle Sarcasm Detection:**
    - Model struggles with deadpan sarcasm lacking obvious markers
    - Context-dependent sarcasm requires world knowledge we don't provide
2. **News Headline Specificity:**
    - Model trained on news-style text may not generalize to conversational sarcasm
    - Different domains (tweets, reviews, chat) might require retraining
3. **Computational Cost:**
    - Grid search with 5-fold CV is computationally expensive
    - Character-level features significantly increase training time

7.3 Insights About Sarcasm Detection

Linguistic Insights:

1. **Punctuation is crucial**: Excessive exclamation marks, question marks, and quotation marks can be strong sarcasm indicators
2. **Phrase patterns**: Bigrams like "oh great", "yeah right", "sure thing" are highly predictive
3. **Contradictions**: Positive words in negative contexts (or vice versa) indicate sarcasm

Technical Insights:

1. **Feature diversity helps**: Combining different feature types (word, char, handcrafted) more effective than optimizing single type
2. **Ensemble > Single model**: Even small ensemble improvements matter in competitions
3. **Regularization is critical**: Preventing overfitting more important than maximizing training accuracy
4. **Domain matters**: Sarcasm patterns differ across domains (news vs. social media vs. reviews)

7.4 Model Limitations

1. **No Transformer Architecture:**
    - Assignment constraints prevented using BERT, RoBERTa, etc.
    - Transformers would capture longer-range dependencies and context better
2. **No External Context:**
    - Can't access world knowledge needed for some sarcasm
    - Example: "Politicians finally solve climate crisis" requires knowing this didn't happen
3. **Limited Negation Handling:**
    - Basic features don't explicitly model negation patterns
    - "not great" vs "great" handled implicitly through bigrams

7.5 Potential Improvements

1. **Incorporate contextual information:** The current model classifies each text independently. Including conversational or surrounding context could help disambiguate sarcastic statements that rely on prior information.
2. **Improve feature engineering:** Adding features that explicitly model sentiment contrast or expectation violation may reduce errors in subtle sarcasm cases.
3. **Data augmentation:** Generating paraphrased or synthetically modified examples could help improve generalization and robustness across datasets.
4. **Cross-domain evaluation:** Testing and training on data from multiple sources or platforms could improve the model's ability to generalize to unseen domains.

8. Conclusion

8.1 Summary of Findings

This project successfully developed a sarcasm detection system achieving **86.23% accuracy** and **84.90% F1-score** on the test set through a combination of Multi-level feature engineering, ensemble learning, rigorous hyperparameter optimization, and conservative regularization. The

model demonstrates strong performance on news-style text and provides solid predictions through machine learning approaches.

8.2 Key Takeaways

1. **Feature engineering matters**: Thoughtful feature design can achieve strong results
2. **Regularization is crucial**: Preventing overfitting is very important
3. **Ensemble methods work**: Combining complementary models reliably improves performance
4. **Domain understanding helps**: Knowing sarcasm characteristics (punctuation, phrases) guides feature design
5. **Incremental testing**: Ablation studies reveal which components actually contribute
6. **Error analysis**: Understanding failure modes guides improvement efforts
7. **Generalization focus**: Cross-validation and monitoring train-val gaps critical for real-world deployment

9. References

Joshi, Aditya, et al. "Automatic Sarcasm Detection: A Survey." *ArXiv.org*, 2016, arxiv.org/abs/1602.03426.

Karmaker, Subrata. "Sarcasm Detection on Reddit Using Classical Machine Learning and Feature Engineering." *ArXiv.org*, 2025, arxiv.org/abs/2512.04396.

Desmond C. "Statistics and Analytics for the Social and Computing Sciences." *6.3 Examples: Logistic Regression*, desmond-ong.github.io/stats-notes/examples-logistic-regression.html. Accessed 17 Dec. 2025.