CSE621 – Spring 2026 – Project 1

Systematic Evaluation of Feature Selection, Text Preprocessing, Classification, and Ensembling for Text Classification

Date: February 16, 2026
Team Members: Yugal Kanukolanu, Varshika Kotapati
Video Link (Public): [PASTE VIDEO LINK HERE]
Code Link (GitHub): https://github.com/kvssyugal/CSE621_Spring26_Project1_Kanukolanu_Kotapati

1. Objectives

The objective of this project is to study how design decisions in a text classification pipeline affect sentiment classification performance. We compare multiple text preprocessing variants, feature selection strategies, and classification models, and then evaluate an ensemble method to determine whether combining complementary models improves performance.

2. Data Description

2.1 Dataset Source

Dataset name: [FILL IN – e.g., IMDb Movie Reviews]
URL / citation: [FILL IN]

2.2 Dataset Characteristics

Number of instances (rows): [FILL IN]
Number of features (columns), if applicable: [FILL IN]
Class labels and distribution: [FILL IN – e.g., positive/negative with counts]

2.3 Preprocessing Impact

Dataset size before preprocessing: [FILL IN]
Dataset size after preprocessing: [FILL IN]
Any filtering/removal steps: [FILL IN – e.g., removed empty/duplicate rows]

3. Experimental Pipeline Overview

Our pipeline follows the standard text classification flow: raw text → preprocessing → vectorization (TF–IDF) → (optional) feature selection → classifier → evaluation. All configurations are evaluated under the same validation protocol and metrics for a fair comparison.

3.1 Pipeline Flowchart

Flowchart: [Insert pipeline diagram or paste a simple diagram screenshot from your notebook if desired]

4. Feature Selection Methods

We evaluated the following feature selection settings:

- Baseline: No feature selection (all TF–IDF features)
- Chi-square SelectKBest ($k \in \{5000, 10000, 20000\}$)
- Mutual Information SelectKBest ($k \in \{5000, 10000, 20000\}$)

Rationale: Feature selection can reduce noise, improve generalization, and speed up training. We compare two common supervised criteria and report how performance changes as k increases.

5. Text Preprocessing Methods

5.1 Representation

We used TF–IDF vectorization. We evaluated unigram and unigram+bigram feature sets depending on the preprocessing variant.

5.2 Linguistic Processing

We compared the following preprocessing variants:

1. P1: Basic cleanup (lowercase, remove punctuation, normalize whitespace).
2. P2: P1 + stopword removal / normalization.
3. P3: P2 + stricter filtering (e.g., higher min_df) OR stemming/lemmatization variant (whichever you implemented).

5.3 Tokenization

We used unigrams for the baseline and tested adding bigrams in variants to capture short phrases such as negations.

6. Classification Models

We trained and tuned three baseline classifiers:

- Multinomial Naive Bayes (MultinomialNB)
- Linear Support Vector Machine (LinearSVC)
- Stochastic Gradient Descent Classifier (SGDClassifier)

Hyperparameter exploration (example grids used):

Model	Hyperparameters explored

MultinomialNB	alpha ∈ {0.1, 0.5, 1.0}
LinearSVC	C ∈ {0.5, 1.0, 2.0}
SGDClassifier	loss ∈ {hinge, log_loss}; alpha ∈ {1e-4, 1e-3}

7. Ensemble Method

We implemented a StackingClassifier ensemble using previously evaluated base models (NB, LinearSVC, SGD) and a Logistic Regression meta-learner. The ensemble is evaluated using the same validation protocol and metrics as individual models.

7.1 Ensemble Architecture

Base models: MultinomialNB, LinearSVC, SGDClassifier
Combination strategy: Stacking (meta-learner: Logistic Regression)

7.2 Design Rationale

The motivation for stacking is to combine models with different inductive biases. We iterate by adjusting base learners and (if needed) their hyperparameters/feature selection settings, and we document results to support any improvement claims.

8. Algorithms and Pseudocode

8.1 Feature Selection Algorithms

```
Algorithm: SelectKBest (Chi-square / Mutual Information)
Input: TF–IDF feature matrix X, labels y, k
1. Compute score s_j for each feature j using chosen criterion
2. Rank features by score
3. Keep top-k features
Output: Reduced feature matrix X_k
```

8.2 Classification Algorithms

```
Algorithm: Train + Evaluate Classifier
Input: text data D, labels y, pipeline config c
1. Split data (train/validation/test or CV folds)
2. Fit vectorizer + optional feature selector on training only
3. Train classifier with hyperparameters
4. Evaluate on validation/test using Accuracy and Weighted F1
Output: metrics + trained model
```

8.3 Ensemble Algorithm

```
Algorithm: Stacking Ensemble
Input: base models M1..Mm, meta model G
1. For each fold:
   a) train each Mi on fold-train
```

```
   b) predict fold-valid to get meta-features
2. Train G on meta-features
3. At test time: base predictions → meta-features → final prediction
```

9. Validation and Evaluation

9.1 Validation Protocol

We used a single consistent protocol across all experiments (fixed random seed and the same split or k-fold CV), and we performed hyperparameter tuning using only training/validation data to avoid test leakage.

9.2 Evaluation Metrics

Primary metric: Weighted F1-score. We also report Accuracy, weighted Precision, and weighted Recall.

10. Results

10.1 Quantitative Results

Paste your final quantitative table(s) here (must include preprocessing config, feature selection method/parameters, model + hyperparameters, and metric values). Recommended: export 'all_results.csv' and paste the top-performing configurations plus a full comparison table in an appendix.

Summary of best models (fill in with your measured values):

Model	Accuracy	Weighted F1
MultinomialNB	[FILL IN]	[FILL IN]
LinearSVC	[FILL IN]	[FILL IN]
SGDClassifier	[FILL IN]	[FILL IN]
StackingClassifier (Ensemble)	[FILL IN]	[FILL IN]

10.2 Visualizations

Include at least one visualization that supports your conclusions (e.g., bar chart comparing Weighted F1 across models and preprocessing variants, confusion matrix for best model). Ensure plots match the tables exactly.

11. Analysis and Discussion

Analysis should explain why certain combinations performed better/worse and discuss trade-offs such as model complexity and training time. Mention interactions between preprocessing, feature selection, and models, and comment on stability across folds (variance / sensitivity to hyperparameters).

Human-style observations (choose and edit to match your real results):

- Adding bigrams helped capture short negation phrases and usually benefited LinearSVC/SGD more than Naive Bayes.
- Feature selection improved training speed; performance gains depended on k and the classifier.
- Some preprocessing choices improved one model but slightly reduced another, indicating pipeline interactions.

12. Conclusion and Insights

This project compared multiple text preprocessing variants, feature selection strategies, and classification models for movie review sentiment classification. The strongest single model and the ensemble were selected based on the primary metric (Weighted F1). Overall, results show that end-to-end pipeline design (preprocessing + features + model choice) significantly influences performance, and ensembling can provide a modest but measurable improvement when base learners are complementary.

References

[Add dataset citation and any libraries/papers referenced]