Sentiment Analysis of IMDB Movie Reviews: A Machine Learning Approach

Authors: Matías Valenzuela, Catalina Herrera **Date:** December 2024

Abstract

This report presents the development and evaluation of a machine learning model for binary sentiment classification of movie reviews. Using the IMDB dataset of 50,000 reviews, we implemented a TF-IDF vectorization approach combined with Logistic Regression. Through systematic hyperparameter optimization using GridSearchCV, our final model achieved an F1-score of 88.11% on the test set, demonstrating the effectiveness of classical NLP techniques for sentiment analysis tasks.

---

1. Introduction

1.1 Problem Statement

Sentiment analysis is a fundamental Natural Language Processing (NLP) task with wide-ranging applications in business intelligence, customer feedback analysis, and content recommendation systems. The goal is to automatically determine whether a piece of text expresses a positive or negative opinion.

1.2 Task Definition

We formulated this as a **binary text classification problem**:

- **Input:** Raw text of a movie review
- **Output:** Binary label (0 = negative, 1 = positive)
- **Evaluation:** Standard classification metrics (accuracy, precision, recall, F1-score)

1.3 Objectives

1. Implement a complete machine learning pipeline for text classification
2. Apply proper data splitting strategies to prevent overfitting
3. Optimize model performance through systematic hyperparameter tuning
4. Achieve competitive results on a benchmark dataset
5. Provide clear documentation and reproducible code

1.4 Background

---

2. Dataset

2.1 Dataset Description

We utilized the IMDB movie review dataset, a widely-used benchmark in sentiment analysis research:

- **Source:** HuggingFace datasets library (`load_dataset('imdb')`)
- **Total size:** 50,000 reviews
- **Pre-defined splits:**
  - Training: 25,000 reviews
  - Testing: 25,000 reviews
- **Class distribution:** Perfectly balanced (50% positive, 50% negative in both splits)
- **Review characteristics:** Variable length text, natural language with diverse vocabulary

2.2 Data Splitting Strategy

To ensure robust model evaluation and prevent data leakage, we implemented a three-way split:

1. **Training set:** 20,000 reviews (80% of original training data)
   - Used for model training
2. **Validation set:** 5,000 reviews (20% of original training data)
   - Used for hyperparameter tuning and model selection
3. **Test set:** 25,000 reviews (original test split)
   - Used only for final evaluation to provide unbiased performance metrics

The validation split was created using stratified sampling to maintain class balance across all subsets.

---

3. *Methodology*

3.1 Overall Pipeline

Our implementation follows a standard supervised learning pipeline:

Raw Text → TF-IDF Vectorization → Logistic Regression → Prediction

This pipeline was implemented using scikit-learn's Pipeline class, ensuring consistent preprocessing across training and prediction phases.

3.2 Feature Extraction: TF-IDF Vectorization

3.2.1 Rationale

Machine learning algorithms require numerical input, but text is inherently symbolic. We chose **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization because:

- It captures both local (within-document) and global (across-corpus) word importance
- It naturally downweights common words that carry little sentiment information
- It provides sparse, efficient representations suitable for linear models
- It has proven effectiveness in text classification tasks

3.2.2 Technical Implementation

The TF-IDF transformation consists of two components:

1. **Term Frequency (TF):** Measures how often a term appears in a document
2. TF(t,d) = (Number of times term t appears in document d) / (Total terms in document d)
3. **Inverse Document Frequency (IDF):** Measures how rare a term is across all documents
4. IDF(t) = log(Total number of documents / Number of documents containing term t)
5. **TF-IDF Score:**

   TF-IDF(t,d) = TF(t,d) × IDF(t)

Configuration parameters:

- max_features: Limits vocabulary to most frequent terms (reduces dimensionality)
- ngram_range: Defines whether to consider single words, word pairs, or both
- stop_words='english': Removes common words like "the", "is", "and"

3.3 Classification Algorithm: Logistic Regression

3.3.1 Model Selection Rationale

We selected Logistic Regression for several reasons:

1. **Simplicity and interpretability:** Linear decision boundary is easy to understand
2. **Efficiency:** Fast training and prediction, even with high-dimensional data
3. **Strong baseline:** Often competitive with more complex models on text data
4. **Probabilistic output:** Provides confidence estimates, not just class labels
5. **Regularization support:** Built-in L2 regularization prevents overfitting

3.3.2 Mathematical Foundation

Logistic Regression models the probability that a review belongs to the positive class:

$$P(y=1|x) = 1 / (1 + e^{(-z)})$$
where $z = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$

The model learns weights w that maximize the likelihood of the training data.

3.4 Hyperparameter Optimization

3.4.1 Grid Search Strategy

We employed GridSearchCV with 3-fold cross-validation to systematically explore the hyperparameter space. This approach:

- Tests all combinations of specified parameters
- Uses cross-validation to estimate generalization performance
- Selects the configuration with the best validation F1-score
- Prevents overfitting to the validation set

3.4.2 Hyperparameter Grid

We tuned three critical parameters:

Parameter	Tested Values	Description
tfidf__max_features	[10000, 20000]	Vocabulary size
tfidf__ngram_range	[(1,1), (1,2)]	N-gram order
clf__C	[0.5, 1.0, 2.0]	Inverse regularization strength

Total combinations tested: 2 × 2 × 3 = 12 configurations

3.4.3 Optimal Configuration

GridSearchCV identified the following best parameters:

- **max_features = 20000**: Larger vocabulary captures more nuanced sentiment expressions
- **ngram_range = (1, 2)**: Bigrams capture contextual information (e.g., "not good" vs. "very good")
- **C = 2.0**: Moderate regularization balances bias and variance

---

4. Results

4.1 Model Performance Progression

We evaluated three model configurations:

1. **Baseline Model:** Default parameters, no optimization
2. **Best Model (Validation):** After GridSearchCV, evaluated on validation set
3. **Best Model (Test):** Final model evaluated on held-out test set

4.2 Final Test Set Performance

The optimized model achieved the following metrics on the 25,000 unseen test reviews:

Metric	Score
Accuracy	88.10%
Precision	88.08%
Recall	88.14%
F1-Score	88.11%

Interpretation:

- **High accuracy:** Model correctly classifies 88.1% of all reviews
- **Balanced precision/recall:** No significant bias toward either class
- **Strong F1-score:** Confirms robust performance across both positive and negative reviews

4.3 Per-Class Performance

Class	Precision	Recall	F1-Score	Support
Negative (0)	0.88	0.88	0.88	12,500
Positive (1)	0.88	0.88	0.88	12,500

The symmetric performance indicates the model does not favor one sentiment over the other.

4.4 Confusion Matrix Analysis

The confusion matrix reveals:

- **True Negatives:** ~11,000 correctly identified negative reviews
- **True Positives:** ~11,000 correctly identified positive reviews
- **False Positives:** ~1,500 negative reviews misclassified as positive
- **False Negatives:** ~1,500 positive reviews misclassified as negative

The symmetric error distribution confirms balanced model behavior.

4.5 Impact of Hyperparameter Tuning

Comparing baseline vs. optimized model showed measurable improvements:

- Using bigrams (1,2) vs. unigrams (1,1) captured more contextual sentiment
- Larger vocabulary (20K vs. 10K) reduced information loss
- Optimal regularization (C=2.0) prevented both underfitting and overfitting

---

5. Discussion

5.1 Strengths of Our Approach

1. **Methodological rigor:** Proper train/validation/test split prevents optimistic bias
2. **Systematic optimization:** GridSearchCV ensures we explored the parameter space thoroughly
3. **Strong baseline:** Logistic Regression proves highly effective for this task
4. **Reproducibility:** Complete code and clear documentation enable replication

5.2 Model Behavior and Insights

- **N-gram benefits:** Bigrams (word pairs) significantly improved performance by capturing phrases like "not bad", "very good"
- **Vocabulary size matters:** 20K features outperformed 10K, suggesting sentiment is expressed through diverse vocabulary
- **Linear separability:** The success of Logistic Regression indicates that positive and negative reviews are largely linearly separable in TF-IDF space

5.3 Limitations

1. **Linear assumption:** Cannot capture complex non-linear patterns that might exist in text
2. **Bag-of-words approach:** TF-IDF ignores word order beyond bigrams
3. **No semantic understanding:** Treats words as independent tokens, missing deeper meaning
4. **Computational cost:** GridSearchCV with 12 configurations required significant training time

5.4 Potential Improvements

Future work could explore:

- **Deep learning models:** LSTM, GRU, or Transformer architectures (BERT, RoBERTa)
- **Pre-trained embeddings:** Word2Vec, GloVe, or FastText for richer representations
- **Ensemble methods:** Combining multiple models for robust predictions
- **Advanced preprocessing:** Handling negations, sarcasm, and context-dependent sentiment
- **Larger n-grams:** Testing trigrams or higher-order phrases

---

6. Conclusion

This project successfully implemented a sentiment analysis system that achieves 88.11% F1-score on the IMDB benchmark dataset. Key takeaways include:

1. **Proper evaluation methodology is critical:** Our three-way data split ensured unbiased performance estimates
2. **Classical NLP methods remain competitive:** TF-IDF + Logistic Regression provides a strong, interpretable baseline
3. **Hyperparameter tuning matters:** Systematic optimization yielded measurable improvements
4. **Balanced performance:** High scores across both classes indicate a robust, production-ready model

The complete implementation is available in our GitHub repository, including:

- Fully documented Python code (src/train_sentiment_model.py)
- Saved trained model (models/imdb_sentiment_logreg.joblib)
- Confusion matrices and evaluation reports (reports/)
- Dependency specifications (requirements.txt)

Overall, this project demonstrates how a well-engineered classical NLP pipeline can deliver **high accuracy, interpretability, and reproducibility** with relatively low computational cost. The resulting model is robust enough for real-world applications such as opinion monitoring, customer feedback systems, or content moderation, proving that classical ML remains a powerful and accessible solution for modern NLP tasks.

Appendix A: Code Repository

GitHub: https://github.com/MathiCL/analisis-sentimientos

Appendix B: Reproduction Instructions

```
# Clone repository
git clone https://github.com/MathiCL/analisis-sentimientos.git
cd analisis-sentimientos

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Run training pipeline
python src/train_sentiment_model.py
```

Appendix C: Key Dependencies

- Python 3.8+
- scikit-learn 1.0+
- datasets (HuggingFace)
- matplotlib
- joblib