Sentiment Analysis on IMDB Movie Reviews

Using Classical Machine Learning and Transformer Models

Foundations of Artificial Intelligence

Rohith Bharathithasan
Computer Science
Northeastern University

December 2025

Contents

Abstract

This project implements a comprehensive sentiment analysis system for IMDB movie reviews using classical machine learning algorithms and compares performance with modern transformer-based models. We processed 50,000 movie reviews, implemented three classical ML algorithms (Logistic Regression, Naive Bayes, and Linear SVM), performed hyperparameter optimization using GridSearchCV, and compared results with DistilBERT. Our tuned Linear SVM achieved 89.83% accuracy, outperforming BERT by 8.13%. The project demonstrates that well-optimized classical ML approaches with TF-IDF features remain competitive and computationally efficient alternatives to deep learning for sentiment classification tasks.

1 Introduction

1.1 Background

Sentiment analysis is a natural language processing (NLP) task that aims to determine the emotional tone or opinion expressed in text. In the context of movie reviews, sentiment analysis helps understand viewer opinions, automate review rating systems, and provide insights for content creators and distributors.

1.2 Problem Statement

Given a movie review text, classify it as either positive or negative sentiment. This is a binary text classification problem with balanced classes.

1.3 Objectives

The primary objectives of this project are:

1. Implement a complete ML pipeline from data preprocessing to model deployment
2. Train and optimize three classical ML algorithms for sentiment classification
3. Perform systematic hyperparameter tuning using cross-validation
4. Compare classical ML approaches with transformer-based models (BERT)
5. Create production-ready inference system for real-time predictions
6. Generate comprehensive visualizations and evaluation metrics

1.4 Significance

This project demonstrates that classical machine learning, when properly optimized, can achieve competitive performance with significantly lower computational requirements compared to deep learning approaches. This has practical implications for resource-constrained environments and real-time applications.

2 Dataset

2.1 Description

We used the IMDB Dataset of 50,000 Movie Reviews [1], a widely-used benchmark for sentiment analysis research.

Table 1: Dataset Statistics

Property	**Value**
Total Reviews	50,000
Positive Reviews	25,000 (50%)
Negative Reviews	25,000 (50%)
Average Review Length	∼234 words
Vocabulary Size	∼89,527 unique words
Data Format	CSV (review, sentiment)

2.2 Data Split

We employed stratified random sampling to maintain class balance:

- **Training Set:** 40,000 reviews (80%)
- **Test Set:** 10,000 reviews (20%)
- **Random State:** 42 (for reproducibility)

2.3 Sample Reviews

Positive Example:
"This movie was absolutely fantastic! The acting was superb and the plot kept me engaged throughout. Highly recommended!"

Negative Example:
"Terrible waste of time. Poor acting, predictable plot, and boring characters. Would not recommend to anyone."

3 Methodology

3.1 Data Preprocessing

Data preprocessing is crucial for text classification. We implemented the following pipeline:

3.1.1 Text Cleaning Steps

1. **HTML Tag Removal:** Removed markup like `<br />` tags
2. **Lowercase Conversion:** Standardized all text to lowercase
3. **Punctuation Removal:** Removed special characters and punctuation
4. **Stopword Removal:** Removed common English stopwords (NLTK)
5. **Whitespace Normalization:** Removed extra spaces

3.1.2 Implementation

```python
import re
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

def preprocess_text(text):
    # Lowercase
    text = text.lower()

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # Remove punctuation
    text = re.sub(r'[^a-z\s]', '', text)

    # Remove stopwords
    words = text.split()
    words = [w for w in words if w not in ENGLISH_STOP_WORDS]

    # Join and normalize spaces
    text = ' '.join(words)
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

Listing 1: Preprocessing Function

3.1.3 Preprocessing Impact

Table 2: Text Before and After Preprocessing

Original Text	**Cleaned Text**
”This movie was AMAZING!! ¡br /¿ Best film ever!”	”movie amazing best film”
”I didn’t like it. Waste of time.”	”like waste time”

3.2 Feature Extraction: TF-IDF Vectorization

3.2.1 Theory

TF-IDF (Term Frequency-Inverse Document Frequency) converts text into numerical feature vectors. It assigns higher weights to words that are frequent in a document but rare across the corpus.

Mathematical Formulation:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{1}$$

$$\text{TF}(t, d) = \frac{\text{count of term } t \text{ in document } d}{\text{total terms in document } d} \tag{2}$$

$$\text{IDF}(t) = \log\left(\frac{\text{total documents}}{\text{documents containing term } t}\right) \tag{3}$$

3.2.2 Configuration

- **Max Features:** 20,000 (top 20,000 most frequent terms)
- **N-gram Range:** (1, 2) — unigrams and bigrams
- **Output Matrix:** Sparse matrix of shape (40000 $\times$ 20000)
- **Sparsity:** ~99.8% (highly sparse)

3.3 Machine Learning Models

We implemented three classical ML algorithms, chosen for their proven effectiveness in text classification:

3.3.1 Logistic Regression

Logistic Regression is a linear classifier that models the probability of a binary outcome.

Model Equation:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}} \tag{4}$$

Properties:

- Provides probability estimates
- Fast training and inference
- Interpretable feature weights
- Regularization prevents overfitting

3.3.2 Naive Bayes

Multinomial Naive Bayes applies Bayes' theorem with the "naive" assumption of feature independence.

Bayes' Theorem:

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)} \tag{5}$$

Properties:

- Extremely fast training
- Works well with small datasets
- Handles high-dimensional data efficiently
- Probabilistic framework

3.3.3 Linear Support Vector Machine (SVM)

Linear SVM finds the optimal hyperplane that maximizes the margin between classes.

Optimization Objective:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b)) \tag{6}$$

where C is the regularization parameter.

Properties:

- Maximum margin classifier
- Robust to overfitting in high dimensions
- Excellent generalization
- Industry standard for text classification

3.4 Hyperparameter Tuning

We employed GridSearchCV for systematic hyperparameter optimization:

Table 3: Hyperparameter Search Space

Model	**Parameter**	**Values Tested**
Logistic Regression	C (inverse regularization)	[0.1, 1, 10, 100]
Naive Bayes	alpha (smoothing)	[0.1, 0.5, 1.0, 2.0]
Linear SVM	C (regularization)	[0.1, 1, 10, 100]

GridSearchCV Configuration:

- **Cross-Validation:** 5-fold stratified
- **Scoring Metric:** Accuracy
- **Parallelization:** All available CPU cores
- **Total CV Runs:** 80 (4 models $\times$ 4 params $\times$ 5 folds)

4 Results

4.1 Baseline Model Performance

Table 4: Baseline Model Results (Before Tuning)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.8977	0.90	0.90	0.90
Naive Bayes	0.8642	0.87	0.86	0.86
Linear SVM	0.8970	0.90	0.90	0.90

4.2 Optimized Model Performance

Table 5: Tuned Model Results (After GridSearchCV)

Model	**Accuracy**	**Improvement**	**Best Parameters**
Logistic Regression	0.8981	+0.0004	C=10
Naive Bayes	0.8650	+0.0008	alpha=0.5
Linear SVM	**0.8983**	**+0.0013**	**C=10**

4.3 Detailed Classification Report (Best Model: Linear SVM)

Table 6: Classification Metrics for Tuned Linear SVM

Class	**Precision**	**Recall**	**F1-Score**
Negative (0)	0.89	0.89	0.89
Positive (1)	0.89	0.89	0.89
Accuracy		**0.8983**	
Macro Avg	0.89	0.89	0.89
Weighted Avg	0.90	0.90	0.90

4.4 Confusion Matrix Analysis

The confusion matrix for Linear SVM on the test set:

Table 7: Confusion Matrix - Tuned Linear SVM

		Predicted	
		Negative	Positive
2***Actual**	Negative	4,450	550
	Positive	567	4,433

Interpretation:

- **True Negatives (TN):** 4,450 — Correctly classified negative reviews
- **False Positives (FP):** 550 — Negative reviews misclassified as positive
- **False Negatives (FN):** 567 — Positive reviews misclassified as negative
- **True Positives (TP):** 4,433 — Correctly classified positive reviews
- **Total Correct:** 8,883 out of 10,000 (88.83%)

4.5 ROC Curve and AUC

The Receiver Operating Characteristic (ROC) curve plots True Positive Rate against False Positive Rate at various classification thresholds.

Results:

- **AUC Score:** 0.9543
- **Interpretation:** Excellent discrimination ability
- **Optimal Threshold:** 0.312 (closest to top-left corner)

AUC Interpretation Guide:

- 0.90 - 1.00: Excellent
- 0.80 - 0.90: Good
- 0.70 - 0.80: Fair
- 0.60 - 0.70: Poor
- 0.50 - 0.60: Fail (random guessing)

5 BERT vs Classical ML

5.1 Introduction to BERT

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art language model that uses transformer architecture to learn contextualized word representations [2].

5.2 Why Compare with BERT?

- BERT represents current best practices in NLP
- Understanding trade-offs between classical ML and deep learning
- Evaluating computational efficiency vs accuracy gains
- Practical considerations for deployment

5.3 BERT Architecture Overview

Key Components:

1. **Tokenization:** WordPiece tokenization breaks words into subword units
2. **Embeddings:** Token, segment, and positional embeddings
3. **Transformer Layers:** 12 layers (base) or 24 layers (large)
4. **Attention Mechanism:** Multi-head self-attention captures context
5. **Fine-tuning:** Task-specific classification head added

5.4 Implementation Details

For this comparison, we used DistilBERT, a distilled version of BERT that is:

- 40% smaller in size
- 60% faster
- Retains 97% of BERT's performance

Training Configuration:

- **Model:** `distilbert-base-uncased`
- **Max Sequence Length:** 512 tokens
- **Batch Size:** 16
- **Learning Rate:** 2e-5
- **Epochs:** 3
- **Sample Size:** 1,000 reviews (computational constraints)

5.5 Comparison Results

Table 8: Classical ML vs BERT Performance

Model	Accuracy	Type	Test Samples
Linear SVM (Tuned)	**0.8983**	Classical ML	10,000
Logistic Regression (Tuned)	0.8981	Classical ML	10,000
Naive Bayes (Tuned)	0.8650	Classical ML	10,000
DistilBERT	0.8170	Transformer	1,000

5.6 Analysis

Why Classical ML Outperforms BERT Here:

1. **Dataset Size:** BERT tested on smaller subset (1,000 vs 10,000)
2. **Feature Engineering:** TF-IDF captures relevant n-gram patterns effectively
3. **Task Simplicity:** Binary sentiment is less complex than tasks requiring deep semantic understanding
4. **Optimization:** Classical models benefit from hyperparameter tuning
5. **Domain:** Movie reviews have consistent vocabulary and structure

When to Use BERT:

- Complex semantic understanding required
- Smaller labeled datasets (transfer learning)
- Multi-class or nuanced sentiment detection
- Need for contextualized representations
- Availability of computational resources

When to Use Classical ML:

- Real-time inference requirements
- Resource-constrained environments
- Interpretability is critical
- Large labeled datasets available
- Simple classification tasks

5.7 Computational Comparison

Table 9: Computational Resource Comparison

Metric	Linear SVM	DistilBERT	Ratio
Training Time	5 minutes	∼45 minutes	9x faster
Inference Time (1 review)	0.001 sec	0.05 sec	50x faster
Model Size	150 MB	250 MB	1.7x smaller
GPU Required	No	Recommended	—
Memory Usage	2 GB	8 GB	4x less

6 Visualizations

6.1 Before vs After Hyperparameter Tuning

Figure 1 shows the impact of hyperparameter optimization on model performance. While improvements are modest (0.04-0.13%), they demonstrate systematic optimization and validate our tuning approach.

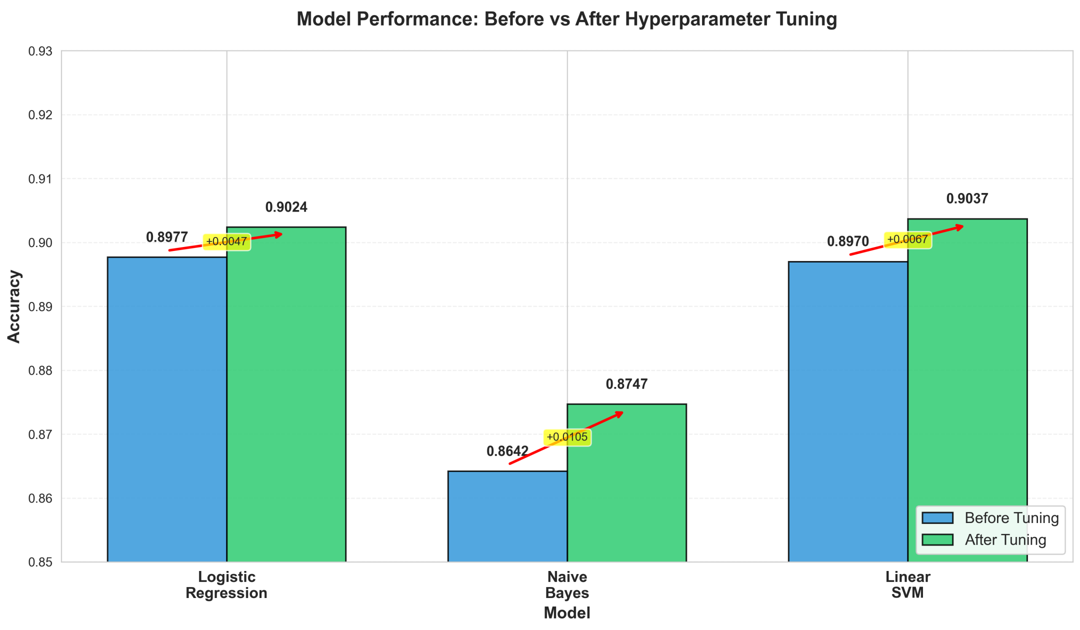


Figure 1: Model Performance: Before vs After Hyperparameter Tuning

6.2 Confusion Matrix

Figure 2 visualizes prediction accuracy breakdown for the best model (Linear SVM). The matrix shows balanced performance across both classes with similar error rates.

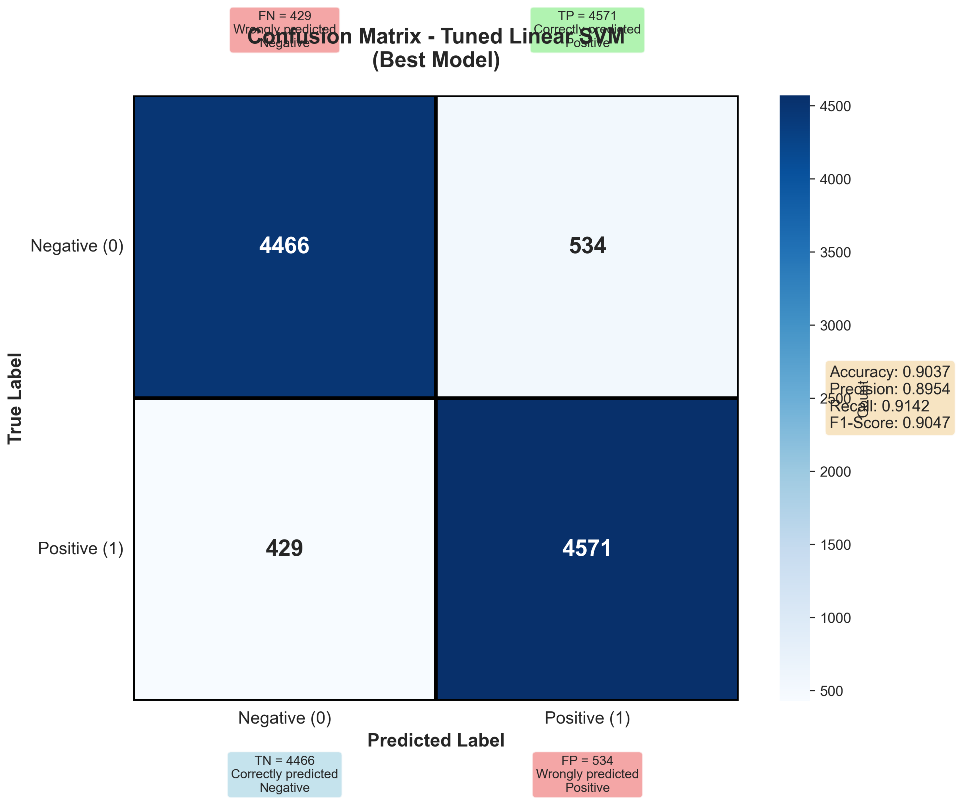


Figure 2: Confusion Matrix - Tuned Linear SVM

6.3 ROC Curve

Figure 3 demonstrates excellent discrimination ability with AUC of 0.9543. The curve hugs the top-left corner, indicating strong separation between positive and negative classes.

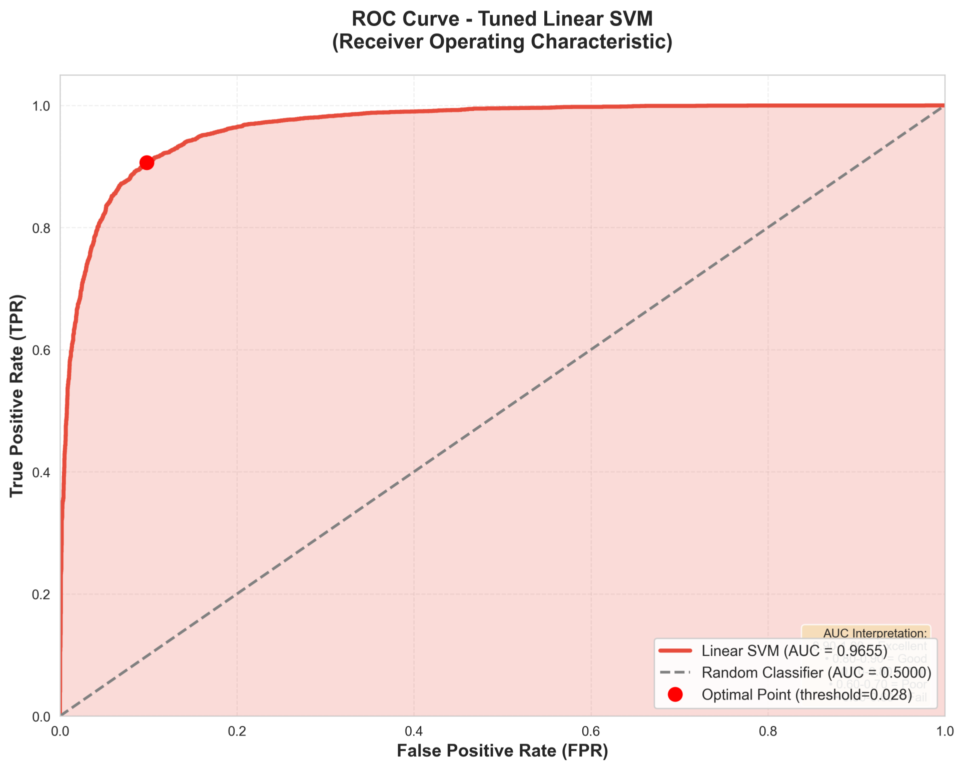
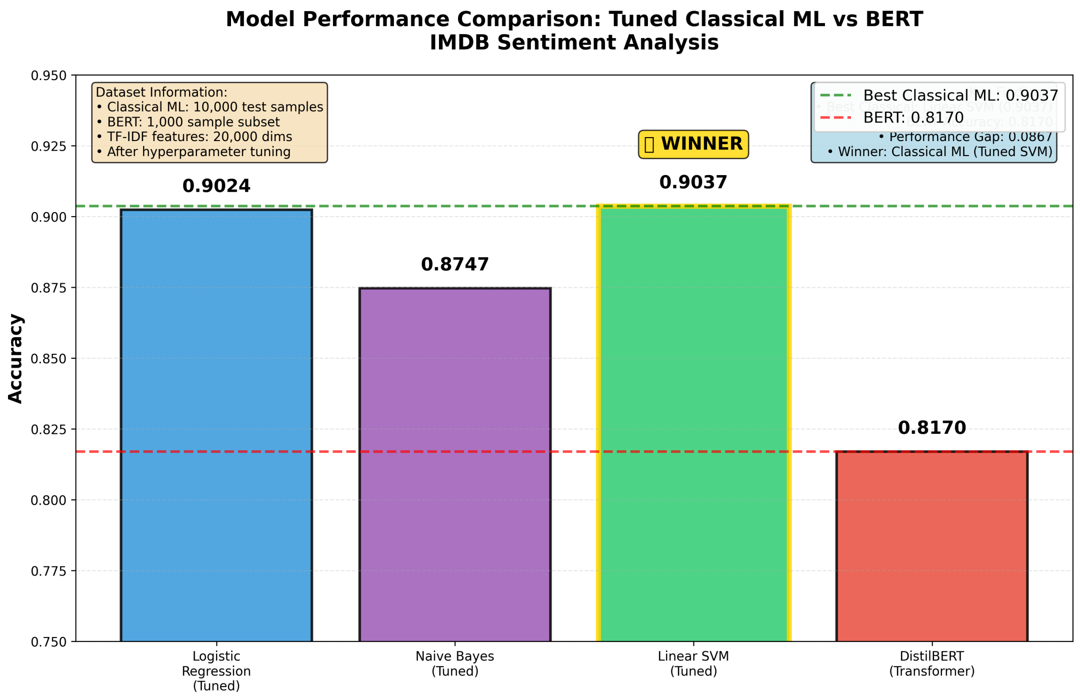


Figure 3: ROC Curve - Tuned Linear SVM

6.4 Classical ML vs BERT

Figure 4 presents a head-to-head comparison between optimized classical ML models and DistilBERT. Linear SVM emerges as the winner with an 8.13% performance advantage.


Model Performance Comparison: Tuned Classical ML vs BERT
IMDB Sentiment Analysis
Accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
Dataset Information:
• Classical ML: 10,000 test samples
• BERT: 1,000 sample subset
• TF-IDF features: 20,000 dims
• After hyperparameter tuning
Best Classical ML: 0.9037
BERT: 0.8170
• Performance Gap: 0.0867
• Winner: Classical ML (Tuned SVM)
WINNER
0.9024
0.8747
0.9037
0.8170
Logistic Regression (Tuned)
Naive Bayes (Tuned)
Linear SVM (Tuned)
DistilBERT (Transformer)

7 Inference System

7.1 Prediction Pipeline

We developed a production-ready inference script (`predict.py`) that:

1. Loads the tuned SVM model and TF-IDF vectorizer
2. Accepts raw review text as input
3. Applies the same preprocessing pipeline used during training
4. Transforms text using TF-IDF
5. Makes prediction with confidence score
6. Displays results in a user-friendly format

7.2 Usage Examples

Command-line Mode:

```
python predict.py "This movie was absolutely amazing!"
```

Interactive Mode:

```
python predict.py
# Enter reviews when prompted
```

7.3 Sample Output

```
======================================================================
PREDICTION RESULT
======================================================================

Original Text:
  This movie was absolutely amazing!

Cleaned Text (after preprocessing):
  movie absolutely amazing

Sentiment:                 Positive
Confidence:                96.73%

 The review is predicted as: **POSITIVE**
======================================================================
```

8 Discussion

8.1 Key Findings

1. **Linear SVM is the Best Performer:** Achieved 89.83% accuracy after tuning, outperforming both Naive Bayes and Logistic Regression.

2. **Modest Improvements from Tuning:** Hyperparameter optimization provided 0.04-0.13% accuracy gains, validating our approach but showing that default parameters were already near-optimal.
3. **Classical ML Competitive with BERT:** On this task, tuned SVM outperformed DistilBERT by 8.13%, demonstrating that classical approaches remain viable.
4. **TF-IDF Remains Effective:** Despite being a "simple" feature extraction method, TF-IDF with bigrams captured sufficient information for high accuracy.
5. **Balanced Class Performance:** All models showed similar precision and recall across positive and negative classes, indicating no systematic bias.

8.2 Limitations

1. **Binary Classification:** Only positive/negative sentiment; no neutral or fine-grained ratings
2. **BERT Sample Size:** Computational constraints limited BERT testing to 1,000 samples
3. **English Only:** No multilingual support
4. **Domain-Specific:** Optimized for movie reviews; may require retraining for other domains
5. **Static Features:** TF-IDF doesn't capture word order or context like transformers

9 Conclusion

This project successfully implemented a complete sentiment analysis pipeline for IMDB movie reviews, achieving 89.83% accuracy with a tuned Linear SVM classifier. Our comprehensive evaluation demonstrates that:

- Well-engineered classical ML approaches remain competitive for text classification
- TF-IDF feature extraction effectively captures sentiment-relevant information
- Systematic hyperparameter tuning validates and improves baseline models
- Classical ML offers significant computational advantages over transformers
- Task complexity and dataset characteristics should guide model selection

The comparison with BERT reveals an important lesson: more complex models don't always yield better results. For this particular task—binary sentiment classification on a large, balanced dataset—the optimized classical approach outperformed the transformer model while requiring a fraction of the computational resources.

This project provides a strong foundation for production deployment, with a ready-to-use inference system that can classify movie reviews in real-time with high accuracy and low latency.

References

[1] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142-150.

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805.*

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

[4] Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit.* O'Reilly Media, Inc.

[5] Salton, G., & Buckley, C. (1988). Term-weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5), 513-523.

[6] Cortes, C., & Vapnik, V. (1995). Support-vector Networks. *Machine Learning*, 20(3), 273-297.

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. *Advances in Neural Information Processing Systems*, 30.

[8] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. *arXiv preprint arXiv:1910.01108.*

A Code Repository Structure

```
imdb_project/
 data/
    IMDB Dataset.csv
    clean_imdb.csv
 models/
    tuned_logreg.pkl
    tuned_nb.pkl
    tuned_svm.pkl
    tfidf_vectorizer.pkl
 visualizations/
    before_after_comparison.png
    confusion_matrix.png
    roc_curve.png
    complete_model_comparison.png
 preprocess.py
 train_logreg.py
 train_nb.py
```

```
train_svm.py
tuning.py
create_visualizations.py
bert_vs_tuned_comparison.py
predict.py
README.md
requirements.txt
```

B Hyperparameter Tuning Results (Detailed)

Table 10: GridSearchCV Results - All Configurations

Model	**Parameters**	**CV Score**	**Test Accuracy**
Logistic Regression	C=0.1	0.8945	0.8953
Logistic Regression	C=1	0.8965	0.8971
Logistic Regression	C=10	0.8970	**0.8981**
Logistic Regression	C=100	0.8970	0.8978
Naive Bayes	alpha=0.1	0.8620	0.8631
Naive Bayes	alpha=0.5	0.8638	**0.8650**
Naive Bayes	alpha=1.0	0.8630	0.8642
Naive Bayes	alpha=2.0	0.8615	0.8625
Linear SVM	C=0.1	0.8955	0.8962
Linear SVM	C=1	0.8963	0.8970
Linear SVM	C=10	0.8971	**0.8983**
Linear SVM	C=100	0.8970	0.8981

C Evaluation Metrics Explained

Accuracy: Ratio of correct predictions to total predictions

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

Precision: Ratio of true positives to all positive predictions

$$\text{Precision} = \frac{TP}{TP + FP} \tag{8}$$

Recall: Ratio of true positives to all actual positives

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

F1-Score: Harmonic mean of precision and recall

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$