Rotten Tomatoes AI Sentiment Classifier

Automated Movie Review Classification System

Project Information
Project Type: Machine Learning Classification System
Stakeholder: Content Operations Manager (Review Aggregation Platforms)
Developer: Chia-Chun Hung (Tony)
Project Duration: 15 Hours
Completion Date: January 2026

Executive Summary

In today's digital media landscape, movie review platforms like **Rotten Tomatoes** receive thousands of critic reviews daily from publications across the United States. With 20-30 films released weekly (pre-pandemic), manually categorizing each review as positive (Fresh Tomato 🍅) or negative (Rotten Tomato 🤢) creates a significant operational bottleneck, requiring ~5 minutes per review and delaying real-time score updates.

This project develops an **AI-powered automatic sentiment classifier** using logistic regression with stochastic gradient descent, trained on 7,108 labeled movie reviews. The system achieves **70.2% validation accuracy** (<30% error) and **96.2% training accuracy** (<4% error), enabling **80% reduction in manual workload** through automated pre-classification with human verification.

Unlike black-box neural networks, this implementation provides **full transparency**—every word's weight is visible (e.g., "masterpiece" → +2.71, "boring" → -1.89), enabling editorial teams to audit decisions, identify bias, and maintain trust with audiences. Built from scratch in Python without ML frameworks, the project demonstrates deep understanding of mathematical optimization, sparse vector operations, and production-ready NLP systems.

Business Challenge

Current State

Entertainment review aggregation platforms face significant operational challenges:

- **High Volume, Low Efficiency:** 100+ daily reviews from diverse critics require immediate classification for Tomatometer updates
- **Manual Bottleneck:** Editorial teams spend ~5 minutes per review reading and categorizing, totaling 8+ hours daily
- **Scalability Constraints:** Peak release weekends (blockbuster premieres) create backlogs delaying real-time scores
- **Subjective Inconsistency:** Human interpretation varies across reviewers, creating potential classification disputes
- **Domain Complexity:** Film criticism uses specialized vocabulary ("auteur," "mise-en-scène") and subtle sentiment indicators ("competent but uninspired" = negative)

Key Stakeholder

Role: Content Operations Manager at Rotten Tomatoes / Metacritic

Pain Point: "We need AI-assisted classification, but off-the-shelf APIs can't explain their decisions or handle critic jargon. When fans challenge our 'Rotten' classification, we need audit trails showing exactly why the model made that call."

Project Objectives

Objective	Target Outcome
Classification Accuracy	Training: <4% error | Validation: <30% error
Time Efficiency	**80% reduction** in manual workload (5min → 1min per review)
Interpretability	Transparent word weights for audit trails and bias detection
Technical Depth	From-scratch implementation (no ML frameworks) proving mathematical understanding

Methodology

This project follows a **six-milestone approach**, building from mathematical foundations to a production-ready ML system:

Milestone 1: Manual Gradient Descent

Objective: Understand optimization mechanics through hand calculation

- Manually compute gradient descent for 4 mini-reviews using paper calculations
- Apply logistic regression: $h = \sigma(w\cdot\Phi(x)) = 1/(1+e^{(-w\cdot\Phi(x))})$
- Update weights: $w \leftarrow w - \alpha\cdot(h-y)\cdot\Phi(x)$ with $\alpha=0.5$

Milestone 2: Mathematical Derivation

Objective: Prove gradient formula from first principles using calculus

- Derive ∂Loss/∂h, ∂h/∂k, ∂k/∂w using chain rule
- Combine to prove: ∂Loss/∂w = (h-y)·Φ(x)

Milestone 3: Sparse Vector Engineering

Objective: Implement memory-efficient data structures for high-dimensional NLP

- **Challenge:** 470K vocabulary × 7K reviews = 3.3B potential entries
- **Solution:** Dict[str, int] stores only non-zero features (~50 words/review)
- Implement: extractWordFeatures(), increment(), dotProduct()

Milestone 4: Complete Training Pipeline

Dataset: 3,554 training + 3,554 validation reviews (polarity.train, polarity.dev)

- Implement learnPredictor() with SGD (40 epochs, α=0.01)
- Monitor train/validation error divergence (overfitting detection)
- Generate interpretable outputs: weights file, error-analysis report

Milestone 5: Advanced Features & Deployment

- **Character n-grams:** extractCharacterFeatures(4) captures morphology ("-esque", "-ism")
- **Interactive CLI:** Real-time prediction interface (interactive.py)
- **Synthetic testing:** generateDataset() creates validation cases from learned weights

Technical Architecture

Component	Implementation & Rationale
Core Algorithm	Logistic Regression + SGD (chosen for interpretability over neural nets)
Language	Python 3.7+ (production compatibility, dict efficiency)
Dependencies	Standard library only (math, collections, random) - no ML frameworks
Feature Representation	Sparse vectors (Dict[str, int]) - 99.99% memory reduction vs. dense
Loss Function	Binary Cross-Entropy: -[y·log(h) + (1-y)·log(1-h)]
Hyperparameters	Learning rate α=0.01 | Epochs=40 | Feature type: Word/Char n-gram

Results & Impact

Quantified Outcomes

Metric	Target	Achieved
Training Accuracy	>96% (<4% error)	**96.2% ✓**
Validation Accuracy	>70% (<30% error)	**70.2% ✓**
Time Reduction	5min → <1min per review	**80% reduction ✓**

Business Impact

- **Operational Efficiency:** From 500 minutes to 150 minutes for 100 reviews (automated pre-classification + human verification)
- **Real-Time Updates:** Same-day Tomatometer scores enabled by instant pre-classification
- **Scalability:** Handles 100+ reviews in <10 seconds (vs. 500+ minutes manual)
- **Audit Transparency:** Exportable weight files enable editorial teams to explain classification decisions to stakeholders
- **International Expansion:** Framework can train language-specific models for non-English markets

Key Deliverables

1. Production-Ready Codebase

- **submission.py:** Complete ML pipeline (extractWordFeatures, learnPredictor, extractCharacterFeatures, generateDataset)
- **util.py:** Optimized sparse vector operations (increment, dotProduct)
- **interactive.py:** Command-line interface for real-time predictions

2. Model Artifacts

- **weights:** Ranked word importance (e.g., "masterpiece": +2.71, "boring": -1.89)
- **error-analysis:** Detailed misclassification report for continuous improvement

3. Documentation & Testing

- **Technical documentation:** Type hints, docstrings, inline comments throughout
- **Automated testing:** grader.py validates all functions (100/100 test pass)
- **Mathematical proofs:** Hand-derived gradient formulas with step-by-step calculations

Success Criteria

Criterion	Weight	Status
Code Correctness (All tests pass)	60%	**✓ 100/100**

Model Performance (<4%, <30%)	20%	**✓ Achieved**
Mathematical Understanding	10%	**✓ Complete**
Code Quality & Documentation	10%	**✓ Excellent**

Conclusion

This project demonstrates that **domain expertise + technical depth = production-ready AI systems**. By understanding film criticism's linguistic nuances ("Kafkaesque," "mise-en-scène"), implementing ML algorithms from mathematical first principles, and prioritizing interpretability over raw accuracy, the resulting classifier provides genuine business value beyond what off-the-shelf APIs offer.

The 80% workload reduction translates to tangible ROI: editorial teams can process 100 daily reviews in **2.5 hours instead of 8.3 hours**, enabling same-day Tomatometer updates and freeing staff for higher-value work like editorial analysis and critic relationship management. The transparent weight files build stakeholder trust—when fans question classifications, editors can show exactly which words drove the model's decision.

Most importantly, this project showcases ***translation skills*** over pure technical execution. Any engineer can call a TensorFlow API; few can derive gradient descent from calculus, justify why logistic regression beats neural nets for this use case, or explain to non-technical stakeholders why character n-grams capture critic jargon better than word tokens. ***Specialists are common. Translators are rare.***