Sentiment Analysis Project Report

Project Problem Description

Problem Definition

Sentiment analysis is the task of determining the emotional tone behind a body of text, classifying it as positive, negative, or neutral. This project focuses on binary classification (positive vs. negative) for movie reviews, useful for applications like recommendation systems, market analysis, or content moderation.

Objectives

- Build an end-to-end ML system with training, serving, and a user interface.
- Use a real-world dataset for practical demonstration.
- Achieve good accuracy on movie review sentiment classification.

Dataset Details

The dataset used is the IMDB Dataset of 50K Movie Reviews from Kaggle. - Source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews/data - Size: 50,000 samples (25,000 positive, 25,000 negative). - Features: 'review' (text column). - Labels: 'sentiment' ('positive' or 'negative', mapped to 1/0). - Why it matters: Movie reviews reflect public opinion, helping studios understand audience reception, predict box office success, and improve content. Sentiment analysis automates this at scale.

Solution Overview

Step-by-Step Approach

1. **Data Preprocessing**:
   - Load CSV using pandas from `data/`.
   - Map labels to binary (1=positive, 0=negative).
   - Use CountVectorizer (limited to 5000 features for efficiency) to convert text to bag-of-words features.
   - Split data into train/test sets (80/20 split).
2. **Model Architecture**:
   - Logistic Regression classifier (simple, interpretable, and effective for high-dimensional text data).
   - Trained using scikit-learn's implementation with max_iter=200 for convergence.
3. **Training Process**:
   - Fit the vectorizer and model on training data.
   - Evaluate on test data (accuracy printed during training, expected ~88%).
   - Save model and vectorizer using joblib to `models/`.
4. **Serving**:

- Load saved model in `src/app.py`.
- Use Gradio to create a web interface: Input review text, output predicted sentiment.

Example Workflow

- Download dataset to `data/` → Run `src/train.py` → Models saved to `models/`.
- Run `src/app.py` → Gradio app launches for predictions.

Challenges and Learnings

Challenges

- Large dataset (50k samples) increases training time; limited features to 5000 to manage memory and speed.
- Text data issues like HTML tags or noise in reviews; basic vectorizer handles it but advanced cleaning (e.g., removing stop words) could improve.
- Imbalanced nuances (dataset is balanced, but real data might not be).
- No deep learning due to simplicity; BERT could achieve >90% accuracy but requires GPU.

How Overcome

- Used efficient scikit-learn implementation.
- Added max_features to vectorizer.
- Assumed basic preprocessing; users can extend with NLTK for stemming/lemmatization.

Key Takeaways

- Real datasets like IMDB provide better generalization than toy data.
- End-to-end ML involves data handling (pandas), modeling (scikit-learn), and deployment (Gradio).
- Trade-offs: Simple models are fast and explainable; scale to transformers for SOTA performance.
- Professional structure separates concerns (code, data, models, docs) for maintainability.

References

- scikit-learn: https://scikit-learn.org/ (for LogisticRegression and CountVectorizer).
- Gradio: https://gradio.app/ (for frontend).
- Joblib: https://joblib.readthedocs.io/ (for model persistence).
- Pandas: https://pandas.pydata.org/ (for data loading).
- Dataset: IMDB Dataset of 50K Movie Reviews on Kaggle.

- Tutorials: scikit-learn text classification guide, Kaggle notebooks on IMDB sentiment.