# P15: Sentiment Classification of IMDb Movie Reviews

CS485 – Project Report

Zervos Spiridon Chrisovalantis (csd4878)
Drakakis Rafail (csd5310)

May 26, 2025

**Abstract**

We present an end-to-end pipeline for sentiment classification of the IMDb Movie Reviews dataset using both classical and deep learning techniques. We preprocess the data, extract features, train multiple models, and evaluate their performance. Classical ML models include Logistic Regression, Naive Bayes, and Linear SVM trained on TF–IDF vectors. Deep learning models include LSTM and 1D-CNN built using PyTorch with learned embeddings. We analyze accuracy, confusion matrices, and inference time. Our results show strong performance from classical methods, competitive CNN results, and poor LSTM learning without pretrained embeddings or tuning.

# Contents

**1 Introduction** **3**

**2 Dataset** **3**

**3 Preprocessing** **3**

**4 Feature Engineering** **3**

**5 Model Architectures & Training Details** **3**
5.1 Classical Models . . . . . 3
5.2 Deep Learning Models . . . . . 4

**6 Implementation Details** **4**
6.1 Preprocessing & Vocabulary . . . . . 4
6.2 Feature Extraction . . . . . 4
6.3 Prediction Script . . . . . 4

**7 Results** **5**
7.1 Classical Machine Learning . . . . . 5
7.2 Deep Learning . . . . . 6

**8 Discussion** **6**

**9 Conclusion** **7**

# 1 Introduction

Sentiment analysis seeks to classify text into categories such as positive or negative sentiment. The IMDb dataset offers a widely used benchmark of 50,000 labeled movie reviews. We compare two families of sentiment classification models:

- **Classical ML**: Feature engineering with TF–IDF followed by Logistic Regression, Naive Bayes, and Linear SVM.
- **Deep Learning**: LSTM and CNN architectures with learned word embeddings and end-to-end training in PyTorch.

# 2 Dataset

- **Name:** IMDb Large Movie Review Dataset
- **Size:** 25,000 training and 25,000 test samples
- **Labels:** Binary sentiment – Positive (1), Negative (0)
- **Source:** `https://ai.stanford.edu/~amaas/data/sentiment/`

# 3 Preprocessing

- Downloaded and extracted from Google Drive.
- Text normalization: lowercasing, tokenization (using NLTK `word_tokenize`), removal of punctuation, numbers, and stopwords.
- For deep models: vocabulary built with a minimum frequency of 2 and capped at 20,000 tokens. Input sequences are padded or truncated to a maximum length of 200.

# 4 Feature Engineering

- **Classical ML:** TF–IDF vectorization with unigrams and 5,000 max features.
- **Deep Learning:** Trainable embedding layers of dimension 100.

# 5 Model Architectures & Training Details

## 5.1 Classical Models

**Logistic Regression:** $\ell_2$ regularization, $C = 1.0$, max_iter = 1000

**Multinomial Naive Bayes:** $\alpha = 1.0$

**Linear SVM:** $C = 1.0$

### 5.2 Deep Learning Models

- **Embedding Layer:** Embedding dimension = 100
- **LSTM:** Single-layer LSTM with 128 hidden units; classifier uses final hidden state
- **CNN:** 1D convolutions with filter sizes [3, 4, 5], 100 filters each; followed by max pooling and a fully connected layer
- **Training:** Optimized with Adam, learning rate = $1e^{-3}$, 5 epochs, batch size = 64

## 6 Implementation Details

### 6.1 Preprocessing & Vocabulary

- NLTK downloads handled programmatically for required packages: `punkt`, `punkt_tab`, `stopwords`.
- Texts are lowercased, tokenized (using `word_tokenize`), filtered for alphabetic words, and cleaned of English stopwords.
- For deep models: a vocabulary is built from training tokens with `min_freq=2`, `max_size=20000`. Sequences are padded or truncated to a fixed length of 200.

### 6.2 Feature Extraction

- **Classical ML:** TF–IDF vectorization (unigrams only), limited to the top 5000 features.
- **Deep Learning:** Each word is mapped to a trainable embedding of dimension 100.

### 6.3 Prediction Script

A separate standalone script, `predict.py`, is provided for inference on new review texts using the trained CNN model.

- **Architecture:** Re-implements the exact CNNClassifier architecture used during training.
- **Workflow:**
  1. Loads the vocabulary from `vocab.pkl`.
  2. Loads the trained model weights from `model.pt`.
  3. Preprocesses the input review with the same tokenization and stopword removal.
  4. Converts the token sequence to indices and pads to length 200.
  5. Performs inference using PyTorch (CPU or CUDA).
  6. Outputs the predicted label (positive or negative) and the confidence score.

# 7 Results

## 7.1 Classical Machine Learning

Table 1: Accuracy and Inference Time (ms/sample) on 25,000 test samples

| Model | Accuracy | Time (ms/sample) |
|---|---|---|
| Logistic Regression | 0.880 | 0.47 |
| Multinomial Naïve Bayes | 0.840 | 0.13 |
| Linear SVM | 0.863 | 0.65 |

Table 2: Classification Report for Logistic Regression

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Negative | 0.88 | 0.88 | 0.88 | 12500 |
| Positive | 0.88 | 0.88 | 0.88 | 12500 |
| **Overall accuracy:** 0.88 | | | | |

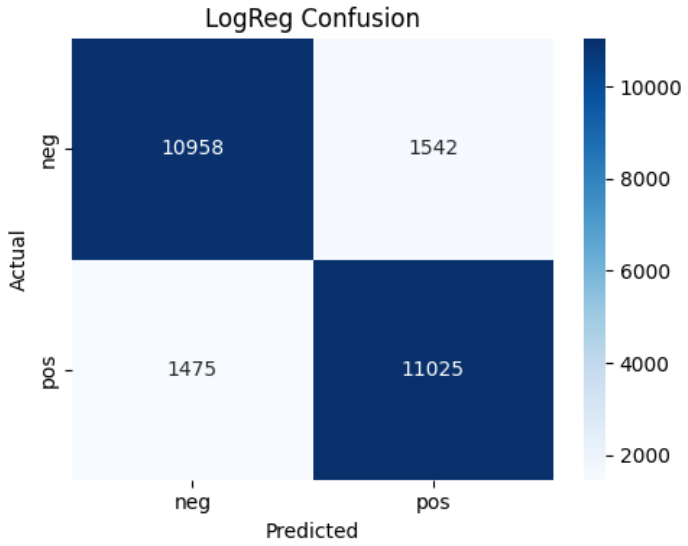


Figure 1: Confusion matrix for Logistic Regression

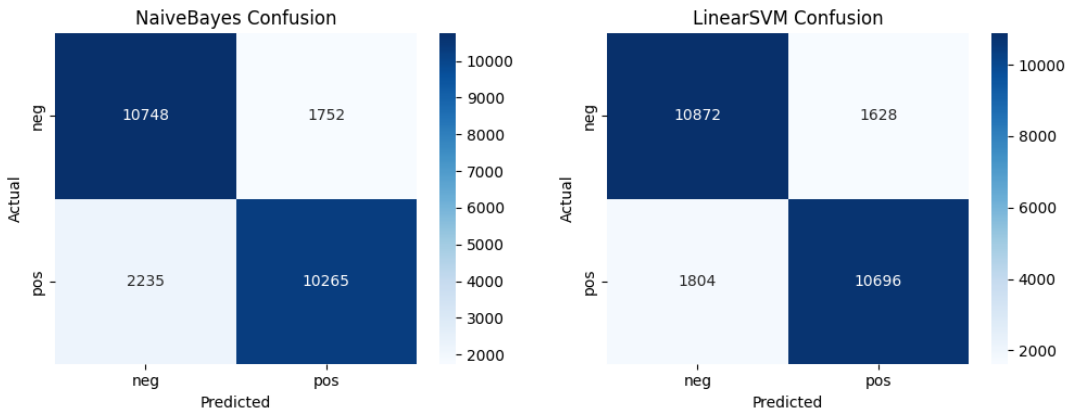


Figure 2: Confusion matrices: Naïve Bayes (left), Linear SVM (right)

## 7.2 Deep Learning

Table 3: Test Accuracy for Deep Models

| Model | Accuracy | Notes |
|---|---|---|
| LSTM | 0.511 | Underfitting, poor learning across epochs |
| CNN | 0.856 | Competitive with classical models |

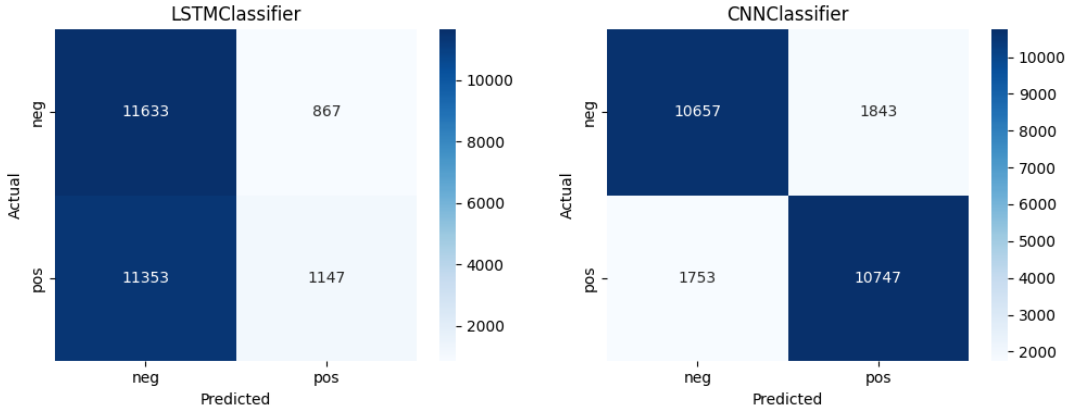


Figure 3: Confusion matrices: LSTM (left), CNN (right)

# 8 Discussion

- **Inference speed:** Naïve Bayes is fastest (0.13ms/sample), followed by Logistic Regression (0.47ms), and Linear SVM (0.65ms).
- **Accuracy:** Logistic Regression is best overall (0.88), CNN comes close (0.856), and LSTM performs poorly (0.511).
- **Deep Learning Issues:**

  - LSTM struggles due to lack of pretraining, insufficient data augmentation, and shallow architecture.
  - CNN benefits from local n-gram pattern detection via convolution and max pooling.

- **Generalization:** CNN is more robust than LSTM but needs GPU for fast training.
- **Short reviews:** Posed challenges for all models, especially with sarcasm or implicit sentiment.

# 9 Conclusion

We implemented a full classification pipeline using classical and deep learning models for sentiment analysis on the IMDb dataset. Classical methods with TF–IDF and Logistic Regression remain strong baselines for accuracy and speed. While the CNN shows promising results, the LSTM architecture underperformed due to training instability and lack of optimization. Further improvements may include hyperparameter tuning, pretrained embeddings, and attention mechanisms.