2025-06-06

# P15: IMDb Sentiment Classification

P15: Sentiment Classification of IMDb Movie Reviews

CS485 – Project Presentation

Zervos Spiridon Chrisovalantis (csd4878)
Drakakis Rafail (csd5310)

University of Crete

Hello everyone, thank you for being here. This is our project presentation for CS485 — we worked on IMDb sentiment classification using both classical and deep learning models. Let's walk you through our process, results, and key takeaways.

Project Objectives

- Build a sentiment classification pipeline for IMDb reviews
- Apply classical and deep learning models
- Evaluate performance: accuracy, speed, and confusion matrices
- Analyze model behavior and limitations

Our goal was to build an end-to-end pipeline that classifies IMDb reviews as positive or negative. We applied classical machine learning models and neural network architectures, and compared them based on accuracy and inference time. We also performed qualitative error analysis to understand where the models fail.

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

We used the IMDb Large Movie Review dataset, a popular benchmark in sentiment analysis. It has 50,000 reviews, evenly split into train and test sets. Each review is labeled as either positive or negative.

Text Preprocessing

- Lowercasing, tokenization using NLTK
- Removing punctuation, numbers, stopwords
- For deep learning:
  - Vocabulary built (min frequency = 2, max size = 20,000)
  - Sequences padded/truncated to 200 tokens

Text preprocessing involved lowercasing, tokenizing using NLTK, and removing punctuation, numbers, and stopwords. For deep learning models, we built a vocabulary using tokens with at least two occurrences, limiting it to 20,000 words. Each review was truncated or padded to a sequence of 200 tokens.

- **Classical ML:** TF–IDF vectorization (unigrams, top 5,000 features)
- **Deep Learning:** Trainable embeddings (dimension = 100)

For classical models, we used TF–IDF vectorization, focusing on unigrams and keeping only the top 5,000 features. For deep learning models, we used a trainable embedding layer with 100 dimensions to represent each word.

- **Logistic Regression:** L2 regularization, $C = 1.0$
- **Naïve Bayes:** Multinomial, $\alpha = 1.0$
- **Linear SVM:** $C = 1.0$

We used three classical models: Logistic Regression, Naïve Bayes, and Linear SVM. Each was tuned with basic parameters — L2 regularization for Logistic Regression, smoothing for Naïve Bayes, and margin penalty for SVM.

Deep Learning Architectures

**LSTM:**

- Single layer, 128 hidden units
- Uses final hidden state

**CNN:**

- Filters: sizes [3, 4, 5], 100 each
- Max pooling and FC layer

The LSTM model had 128 hidden units and used the final hidden state to make a prediction. The CNN model used filters of size 3, 4, and 5 — each with 100 channels. These filters detect n-gram patterns and are followed by max pooling and a dense layer.

## Training Setup

### Training Setup

- Optimizer: Adam, learning rate = $1e^{-3}$
- Epochs: 5, Batch size: 64
- Implemented in PyTorch
- Inference script: `predict.py` (Can be used to predict new reviews)

All models were trained using the Adam optimizer with a learning rate of 0.001. We trained for 5 epochs with a batch size of 64. The CNN model can also be used for inference using our standalone ‘predict.py‘ script.

Classical Model Results

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

TF–IDF features, evaluated on 25k test samples

Logistic Regression achieved the highest accuracy at 88Naïve Bayes was the fastest in terms of inference time — only 0.13 milliseconds per sample. SVM performed slightly worse than Logistic Regression in both accuracy and speed.

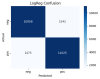
2025-06-06

# P15: IMDb Sentiment Classification

└─Logistic Regression Report

## Logistic Regression Report

- Precision, Recall, F1-score: 0.88 for both classes
- Balanced performance across sentiment labels
- Fast inference: 0.47 ms/sample

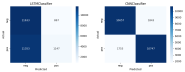



Confusion Matrix: Logistic Regression

The Logistic Regression model had balanced performance — 0.88 precision, recall, and F1-score for both positive and negative classes. The confusion matrix confirms this with very few misclassifications.

## Deep Learning Results

| Model | Accuracy | Notes |
|---|---|---|
| LSTM | 0.511 | Underfitting, unstable learning |
| CNN | 0.856 | Competitive, robust features |




Confusion Matrices: LSTM (left), CNN (right)

The LSTM model performed poorly — around 51This is likely due to shallow architecture and no pretrained embeddings. The CNN model performed well, reaching 85.6

Model Comparison

- **Best Accuracy:** Logistic Regression (0.88)
- **Best DL Model:** CNN (0.856)
- **Fastest Inference:** Naïve Bayes (0.13 ms/sample)
- **Underperformer:** LSTM (0.511), due to lack of tuning and pretraining

To summarize, Logistic Regression had the highest accuracy. CNN was the best deep model. Naïve Bayes was the fastest. LSTM underperformed due to poor generalization and lack of pretrained word embeddings.

Error Analysis

- **LSTM:** Poor generalization without pretrained embeddings
- **CNN:** Stronger with local patterns (n-grams)
- **All Models:**
  - Misclassify short or sarcastic reviews
  - Struggle with implicit sentiment

The LSTM's weakness was its inability to generalize well from scratch. CNN learned robust local patterns and performed better. All models struggled with sarcasm, irony, and short ambiguous reviews — which are hard to interpret without deeper context.

- Classical methods remain strong
- CNNs show promise with competitive accuracy
- LSTM requires optimization: embeddings, attention, deeper networks

In conclusion, classical ML models remain very effective for this task. CNNs are promising and could surpass classical methods with more tuning or pretrained word embeddings. LSTMs require more depth, attention mechanisms, and better initialization to be competitive.

Questions?

Thank you for your attention. We're happy to answer any questions you have!