IMDB Sentiment Classification

Mini-Project 2 — DS52 Machine Learning

Ivann Vasic Ludovic Blondeau Esteban Rambaud

June 2026

1 Data and Objectives

Dataset. We used the *IMDB Dataset of 50K Movie Reviews* (Maas et al., 2011), downloaded from Kaggle. It contains 50,000 English-language reviews scraped from IMDb, each labelled as `positive` or `negative`, with a perfectly balanced class distribution (25,000 positive / 25,000 negative) before any cleaning. After removing 418 exact duplicates, our working dataset has **49,582 reviews** — the balance is preserved, so no resampling or class-weight adjustment was needed.

Task. Binary text classification: given a review string, predict its sentiment label (positive = 1, negative = 0). This is a standard NLP benchmark, but the raw text requires substantial preprocessing before any model can consume it.

Difficulties. Three aspects made the problem non-trivial:

- **Variable length.** Review lengths range from a few words to over 2 000 words (median $\approx$ 240 words). Sequence models like LSTM must truncate inputs (we capped at 150 tokens), discarding up to 40% of longer reviews.
- **Noisy raw text.** Reviews contain HTML artefacts (`<br />`), URLs, and special characters that carry no sentiment signal and must be stripped before vectorisation.
- **Semantic complexity.** Negation ("not good"), sarcasm ("what a *masterpiece*..."), and mixed-sentiment paragraphs are common. Bag-of-words models lose word order and struggle with these cases.

2 Models, Preprocessing, and Tools

Preprocessing pipeline

Raw text was cleaned with compiled regular expressions: HTML tags removed, URLs stripped, non-alphabetic characters deleted (apostrophes kept for contractions like *don't*), and everything lowercased. We then built two vectorisations:

- **TF-IDF** (for LR, SVM, MLP): `TfidfVectorizer` with `max_features=20 000`, unigrams+bigrams (`ngram_range=(1,2)`), `sublinear_tf=True`, and a custom 310-word stoplist that *preserves negations* (*not, no, never, nor, none*). Without this, "not good" and "good" would become identical after stopword removal.
- **Integer sequences** (for LSTM, TextCNN): a `CountVectorizer`-derived vocabulary of 15 000 tokens (`min_df=5`), sequences padded/truncated to 150 tokens.

All vectorisers were **fit only on the training set** to prevent data leakage. We split 70 / 15 / 15% (train / val / test) with stratification; the same split was used for all five models.

Models tested

(LR) Logistic Regression — sklearn's `LogisticRegression` on TF-IDF. Our floor: any neural model must beat this to justify the added complexity.

(SVM) LinearSVC + CalibratedClassifierCV — same TF-IDF features, maximum-margin objective instead of cross-entropy. We wrapped `LinearSVC(C=1.0)` with `CalibratedClassifierCV(cv=3)`

to obtain probability estimates for ROC-AUC computation, which `LinearSVC` does not natively produce.

(MLP) 3-layer feedforward network — architecture: Linear(20 000→256) + ReLU + Dropout(0.4), then Linear(256→128) + ReLU + Dropout(0.4), then Linear(128→1). 5.15M parameters, trained for 8 epochs with Adam (lr=1e-3, weight_decay=1e-5). We chose this because TF-IDF produces fixed-size vectors that are natural inputs for a feedforward network, and non-linear layers can capture feature combinations that LR/SVM cannot.

(LSTM) Unidirectional LSTM with learned embeddings — `Embedding(15 000, 64)` → `LSTM(64, 64)` → Dropout(0.4) → `Linear(64, 1)`. 993K parameters, 8 epochs, gradient clipping (max_norm=5.0). We chose LSTM because word order matters for sentiment: the LSTM reads the sequence token by token and can, in principle, learn that "not good" has opposite polarity to "good".

(TextCNN) 1D Convolutional Network (Kim 2014 style) — same embedding layer as LSTM, then three parallel `Conv1d(64, 128, k)` with kernel sizes $k \in \{3, 4, 5\}$, global max-over-time pooling, Dropout(0.4), `Linear(384, 1)`. 1.06M parameters, 8 epochs. CNNs detect the strongest n-gram signal in each filter size simultaneously, making them faster than LSTM on CPU while still capturing local word-order patterns.

Libraries: PyTorch 2.12 (neural models), scikit-learn 1.8 (vectorisers, LR, SVM, metrics), pandas 3.0, numpy 2.4. All training ran on CPU (no GPU available locally).

3 Results and Comparisons

Quantitative results

Table 1: Test-set performance on 7 438 samples (best value in bold).

Model	Accuracy	Precision	Recall	F1-score	Train time
LR (baseline)	90.0%	89/91%	91/89%	90.2%	0.3 s
SVM (LinearSVC)	90.0%	90/90%	90/90%	90.0%	≈5 s
MLP + TF-IDF	**90.5%**	90/91%	91/90%	**90.5%**	131 s
LSTM	84.9%	85/85%	84/86%	85.1%	1 145 s
TextCNN	85.8%	86/86%	86/86%	85.9%	301 s

Precision/Recall columns show neg/pos class values. ROC-AUC: 96.4% (LR), 96.7% (MLP), 91.9% (LSTM), 93.8% (TextCNN).

Training curves

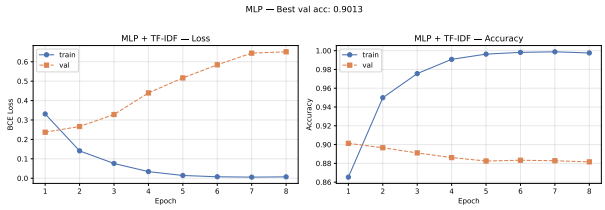
Analysis and honest comparison

The most striking result is that **LR (0.3 s) and SVM (≈5 s) match MLP at 90%** and comfortably beat LSTM (84.9%) despite being 1 000× faster to train. This was genuinely surprising to us: we expected the sequence models to win by capturing negation and context.

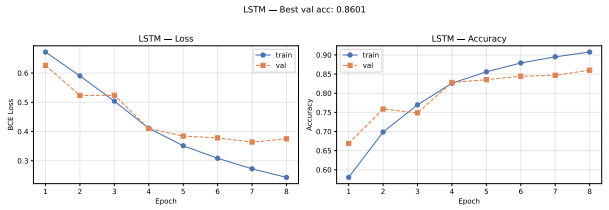
Why the MLP wins overall. The MLP (90.5%) improves over LR/SVM by learning non-linear interactions between TF-IDF features, but the improvement is modest (+0.5%). The TF-IDF bigrams already capture most of the signal — "not good" is a distinct bigram token and does not need word-order modelling.

Why the LSTM underperforms. Figure 1b shows the LSTM starting at only 66.9% validation accuracy at epoch 1, slowly climbing to 86.0% by epoch 8. Learning embeddings from scratch on 50 000 reviews is insufficient: the vectors do not converge to meaningful representations within the available training budget. By contrast, the MLP already achieves 90.1% at epoch 1 (TF-IDF is a strong fixed representation), and its best model is saved from epoch 1 before overfitting sets in (Figure 1a).

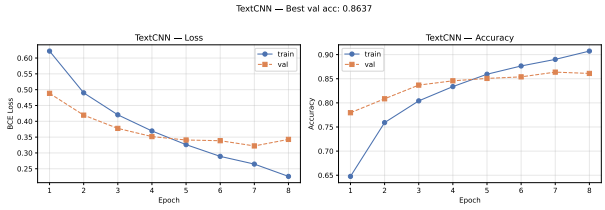
TextCNN vs. LSTM. The TextCNN reaches 85.8% (vs. 84.9% LSTM) in 301 s vs. 1 145 s — a 4×



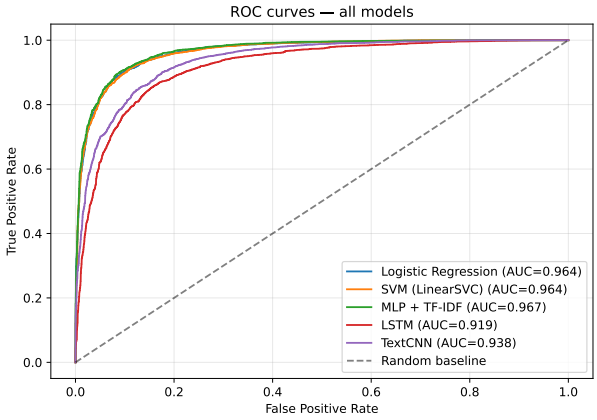
(a) MLP: peaks at epoch 1 (90.1%), then overfits (train → 99%)



(b) LSTM: slow start at 66.9%, gradually climbs to 86.0%



(c) TextCNN: steadier convergence than LSTM, reaches 86.4%



(d) ROC curves — TF-IDF models (solid) clearly dominate

Figure 1: Training (solid) and validation (dashed) curves per epoch (left three), and ROC curves on the test set for all five models (bottom right).

speedup with slightly better accuracy. Its parallel convolutions converge faster because they avoid the vanishing-gradient issues of sequential processing.

SVM vs. LR. Both reach 90.0%, but the SVM produces more balanced per-class precision/recall (90/90% vs. 89/91% for LR), suggesting the maximum-margin objective provides a slightly better decision boundary on this symmetric dataset.

Feature interpretability

Figure 2 reveals *why* TF-IDF performs so well on IMDB: the top positive features are words like *brilliant*, *beautifully*, and bigrams like *the best*, while the top negative features include *awful*, *waste* and bigrams like *worst film* and *not worth*. These are exactly the tokens that unambiguously signal sentiment with no context needed — confirming that bag-of-bigrams already captures most of the discriminative signal, leaving little for recurrence to add.

4 Conclusion and Next Steps

What we learned. This project challenged our assumption that more complex architectures always perform better. LR trained in 0.3 s on a simple bag-of-bigrams outperformed an LSTM that took 19 minutes on CPU. The takeaway is that the *feature representation* matters as much as the model: TF-IDF with preserved negations is already a near-optimal representation for IMDB sentiment, so adding recurrence on top of learned embeddings (which are weak after 50k samples) brings no benefit.

We also learned how to prevent data leakage rigorously: fitting TF-IDF only on the training set, validating on a separate held-out fold, and selecting the best model by validation accuracy rather than training accuracy.

What we would try with more time. The obvious next step is to initialise the `nn.Embedding` layers with pretrained GloVe or BERT vectors. This would likely close most of the gap between LSTM/CNN and MLP, since the embeddings would already encode semantic similarity from the start. We would also fine-tune a small Transformer (DistilBERT) — typically >93% on IMDB — to establish an upper

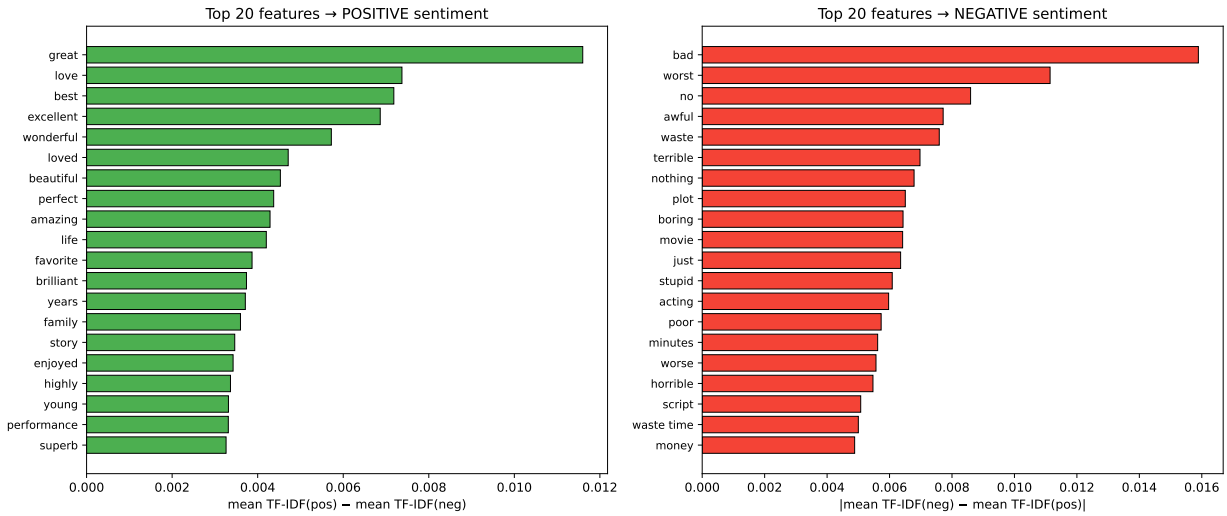


Figure 2: Top 20 most discriminative TF-IDF features per class, ranked by the difference in mean TF-IDF score between positive and negative training reviews.

bound. For the SVM, replacing `LinearSVC` with a kernel SVM (`SVC(kernel='rbf')`) could capture non-linear feature interactions that the linear margin misses, and a grid search over $C \in \{0.1, 1, 10\}$ could squeeze out another 0.5–1%.

What future work could improve. Bidirectional LSTM or attention mechanisms could help focus on the most sentiment-bearing tokens. Handling sarcasm remains an open challenge: our error analysis shows that ironic reviews (e.g., "*what a masterpiece — if you enjoy watching paint dry*") systematically fool all five models.