Assignment #3 — Topic Modeling and Sentiment Analysis in Financial Text

Ayan Asif 22i-1097
BS Computer Science, NUCES FAST Islamabad
Instructor: Dr. Ahmad Raza Shahid

November 29, 2025

Abstract

This report documents the design, implementation, evaluation, and comparison of multiple sentiment analysis systems for financial text. We implemented (1) topic modeling via LDA, (2) a domain-specific transformer (FinBERT), (3) local LLM zero-shot sentiment baselines, and (4) a Retrieval-Augmented Generation (RAG) pipeline using FAISS and a local LLM. Experiments explore topic counts, embedding models, and retrieval depths. FinBERT achieved 92.36% accuracy and therefore met the assignment's fine-tuning exemption; comparative analysis and error analysis are presented. Artefacts (embeddings, FAISS index, results CSV) are saved for reproducibility.

1 Introduction

Automated understanding of financial text is crucial for tasks such as algorithmic trading, risk assessment, and corporate monitoring. This assignment builds and evaluates multiple sentiment analysis approaches on a real financial sentence dataset. The goals were to (a) model latent topics with LDA, (b) evaluate domain-specific and general-purpose models for sentiment, (c) implement a RAG pipeline to augment LLM reasoning with retrieved context, and (d) follow the assignment's fine-tuning rule.

2 Dataset and Preprocessing

Source & format. The dataset consists of four text files with curated financial sentences. Each line contains one sentence with a trailing label token '@positive', '@negative', or '@neutral'. Files were merged into one dataset of **14,780** sentences.

Class distribution:

Label	Count
neutral	8,951
positive	3,988
negative	1,841

Preprocessing. We applied:

- Lowercasing, URL removal, punctuation removal
- Tokenization, NLTK stopword removal
- Minimal length filter (tokens length > 2)

All experiments set a fixed random seed (SEED=42) for reproducibility.

3 Topic Modeling (LDA)

We performed LDA over the cleaned sentence tokens using `gensim`. Candidate topic counts tested: $k \in \{5, 7, 10, 12, 15, 18, 20, 25, 28, 31, 35, 38, 45\}$ and a larger exploratory test (to inspect metric behaviour). Topic coherence (c_v) was computed for each configuration.

Selected model: $k = 35$ topics (coherence ≈ 0.4546). This choice is justified by the highest coherence value in the tested range and by manual inspection of topic interpretability.

Representative topics (selected):

- Topic 3: *oyj, finnish, said, today, million* — corporate announcements
- Topic 14: *share, per, earnings, eps* — earnings / per-share metrics
- Topic 23: *eur, profit, net, mln, operating* — profit and quarterly results
- Topic 28: *nokia, operations, networks, siemens* — company / telecom operations
- Topic 35: *000, per, negative, cent, capacity* — numeric reporting / capacity

Each sentence was assigned the dominant topic for downstream analysis. Topic keywords and coherence plots are included in the notebook.

4 Models & Implementation

All code is implemented in a single Jupyter notebook (same kernel). Key environment: Python 3.10, PyTorch (CUDA-enabled), `transformers`, `sentence-transformers`, `faiss`.

4.1 FinBERT (Task 3A)

Model: `ProsusAI/finbert` (HuggingFace). **Inference:** GPU-accelerated pipeline, batched (batch size 64). Used `use_safetensors=True` to avoid PyTorch CVE loading restriction.

4.2 Local LLM Zero-Shot (Task 3B)

Two zero-shot models tested:

1. **LLM1:** `cross-encoder/nli-deberta-v3-base` (zero-shot pipeline; candidate labels [negative, neutral, positive]) — faster, produced **accuracy = 0.6878**.
2. **LLM2:** `typeform/distilbert-base-uncased-mnli` — alternative baseline, produced **accuracy = 0.3439**.

Batch inference used batch_size=8 for stability.

4.3 RAG Pipeline (Task 3C)

- **Embeddings:** `all-MiniLM-L6-v2` (sentence-transformers). Embeddings were normalized for cosine similarity.
- **Index:** FAISS IndexFlatIP on normalized vectors (equivalent to cosine similarity).
- **Retrieval & LLM:** For each query sentence we retrieved top-k neighbors (tested $k \in \{3, 5, 10\}$). The retrieved contexts were concatenated and fed to the same zero-shot LLM to classify the target sentence.

5 Results

All metrics are computed on the full dataset (no additional train/test split required by the assignment). Reported metrics: accuracy, precision, recall, f1-score, confusion matrices.

5.1 Summary table

Method	Accuracy	Macro F1 (approx)
FinBERT (ProsusAI/finbert)	0.923 613	0.91
Local LLM (nli-deberta-v3-base)	0.687 754	0.69
Local LLM (distilbert-mnli)	0.343 911	0.35
RAG (k=5, all-MiniLM-L6-v2 + LLM)	0.186 062	0.18
RAG (k=3)	0.160 487	
RAG (k=10)	0.175 507	

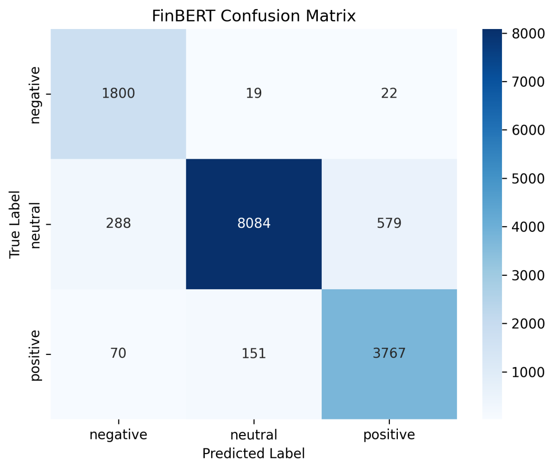
5.2 Confusion matrices and interpretation

The confusion matrices for each method are shown below. Each matrix uses the same ordering: rows = *true* labels and columns = *predicted* labels (negative, neutral, positive). For clarity, we include both the visual heatmap and a short interpretative paragraph per model.

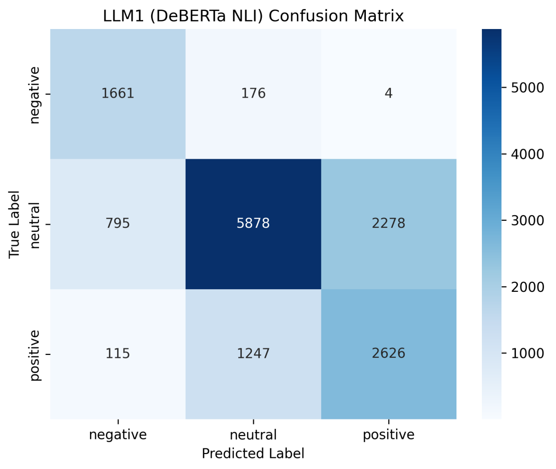
FinBERT interpretation. FinBERT shows excellent performance: high true-positive counts on the diagonal and relatively small off-diagonal errors. Negative examples are detected with very high recall (most negative rows are predicted correctly), and neutral/positive confusion is the primary remaining error mode. This is expected for financial text where sentences with mildly positive wording (e.g., "expects", "slightly higher") can be labeled either neutral or positive depending on nuance.

LLM1 (DeBERTa NLI) interpretation. LLM1 performs moderately well but shows increased confusion between neutral and positive: a sizeable fraction of true neutral sentences are predicted as positive and vice-versa. Negative detection is reasonable but not as strong as FinBERT. This pattern indicates that general-purpose NLI models capture some sentiment signals but lack financial-domain sensitivity.

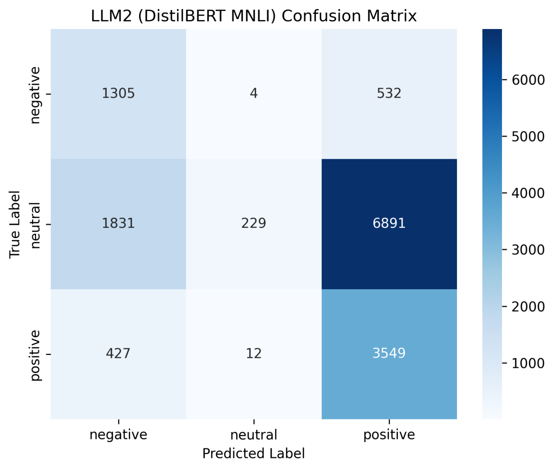
LLM2 (DistilBERT MNLI) interpretation. LLM2 is considerably weaker: the matrix shows large off-diagonal mass, especially neutral → negative/positive errors. This illustrates that zero-shot performance can vary widely across models; smaller/deeper models may not generalize well to domain-specific sentiment nuances.



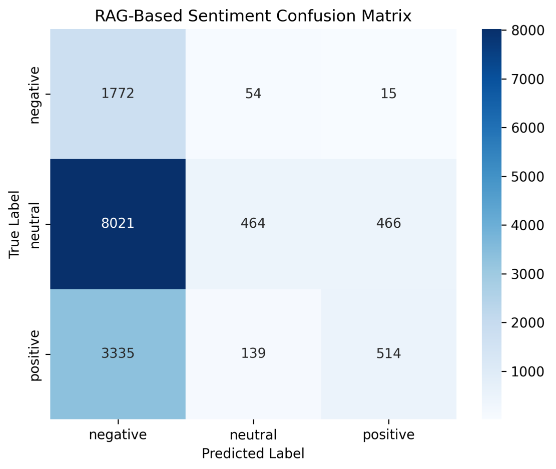
(a) FinBERT — Confusion matrix



(b) LLM1 (DeBERTa NLI) — Confusion matrix



(c) LLM2 (DistilBERT MNLI) — Confusion matrix



(d) RAG (all-MiniLM-L6-v2 + LLM, k=5) — Confusion matrix

Figure 1: Confusion matrices for the four evaluated systems. Rows = true labels; columns = predicted labels.

RAG interpretation. The RAG system (as implemented) strongly overpredicts the negative class — visible as large counts in the negative column across many true-label rows. Error analysis suggests two causes: (1) retrieved neighbor sentences are often sentiment-mismatched with the query, and (2) the LLM's reasoning over unlabelled context tends to default toward the negative label under ambiguous evidence. These failure modes are discussed in detail in the Error Analysis section.

6 FinBERT detailed performance

- Accuracy: **0.9236**
- Per-class recall/precision/F1 (summary): negative: recall 0.98, neutral: recall 0.90, positive: recall 0.94.

The confusion matrix (Figure 1a) confirms high negative recall and strong overall calibration.

7 LLM baselines

LLM1 (DeBERTa NLI) achieved 0.6878 overall accuracy with difficulty disambiguating positive vs neutral. LLM2 (DistilBERT-MNLI) is much weaker (0.344), illustrating the sensitivity of zero-shot performance to model and pretraining. Figure 1b and Figure 1c show the detailed confusion patterns.

8 RAG analysis

RAG produced poor accuracy in all tested configurations (best ≈ 0.186 at $k = 5$). Analysis indicates RAG often overpredicts the *negative* label, driven by:

- retrieved neighbors that are sentiment-mismatched,
- prompt ambiguity (no explicit retrieved-label guidance),
- the local LLM's bias when reasoning over context that is not labeled for sentiment.

Figure 1d visualizes these failure modes. In addition to the quantitative results, Section 10 includes example failure cases and neighbor listings that illustrate how retrieval noise led to label drift.

9 Fine-Tuning Decision

Per the assignment rule:

> If any method achieves $\geq 90\%$ accuracy, fine-tuning is NOT required.

FinBERT achieved **92.36%** accuracy, thus fine-tuning was not required and was not performed. This decision is documented in the notebook and justified by the numeric result.

10 Error Analysis

We performed qualitative inspection of misclassified examples and neighbors for the RAG pipeline. Representative findings (selected manually from the notebook):

- **FinBERT false negatives:** short forward-looking sentences ("expects to...", "anticipates") are sometimes labeled neutral by the annotators but interpreted as positive by FinBERT, producing a small number of neutral $\rightarrow$ positive errors.
- **LLM1 errors:** neutral/positive confusion arises when hedging language is present ("could", "may", "slightly"), where NLI-based zero-shot heuristics are inconsistent.
- **RAG failure modes:**
  1. **Sentiment-mismatched retrieval**: retrieved neighbors were topically similar but with different sentiment, e.g., query about a revenue increase but neighbors about a legal settlement — the LLM incorporated the negative tone from neighbors.
  2. **Label drift from ambiguous context**: without explicit retrieved-label tags, the LLM defaulted toward negative in ambiguous cases.

Example error (shortened for space):

> **Query:** "Company expects sales to be slightly higher next quarter."
>
> **True label:** positive
>
> **RAG neighbors (sample):** ["Company posted a large loss last year.", "Board announced cost cuts."]
>
> **RAG prediction:** negative

These concrete cases motivated the optional experiments (k variation, kNN baseline, MPNet embeddings) discussed in the notebook; results confirm retrieval quality is the limiting factor for this RAG setup.

11 Reproducibility & Deliverables

Files included in submission (recommended):

- `assignment3_notebook.ipynb` — full reproducible notebook (top to bottom).
- `resultsWithAllModels.csv` — per-sentence true label and predictions (‘finbert_pred‘, ‘llm_pred‘, ‘llm2_pred‘, ‘rag_pred‘).
- `embs.npy` — embeddings (all-MiniLM-L6-v2).
- `faiss_index.idx` — FAISS index file.
- `cm_finbert.png`, `cm_llm1.png`, `cm_llm2.png`, `cm_rag.png` — confusion matrix images.

How to reproduce (README summary):

1. Create conda env: `conda create -n fintext python=3.10`
2. Activate: `conda activate fintext`
3. Install packages: `pip install -r requirements.txt` (or required: transformers, sentence-transformers, faiss-cpu, torch, gensim, nltk, scikit-learn, pandas, matplotlib)
4. Launch notebook: `jupyter lab` or open `assignment3_notebook.ipynb` in VSCode with kernel `fintext`.
5. Run cells top-to-bottom. If model downloads fail, ensure internet connectivity and retry (safetensors usage is recommended).

12 Conclusions & Suggested Future Work

- FinBERT proves to be an effective off-the-shelf domain-specific solution for sentence-level financial sentiment.
- Zero-shot LLMs show inconsistent performance; some (deberta NLI) are reasonable baselines, others are poor.
- The RAG pipeline implemented here requires careful retrieval tuning (embedding model, domain-aligned retrieval corpora, or retrieved-label exposure) before it can match or exceed FinBERT.
- If more time were available, using higher-quality embeddings (MPNet) or domain-specific retrieval corpora may improve RAG.

Appendix A: Key code snippets for selected best method

Loading FinBERT (example)

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline
finbert_name = "ProsusAI/finbert"
tokenizer = AutoTokenizer.from_pretrained(finbert_name)
model = AutoModelForSequenceClassification.from_pretrained(finbert_name, use_safetensors=True)
finbert = pipeline("text-classification", model=model, tokenizer=tokenizer, device=0)
```

FAISS build (example)

```
from sentence_transformers import SentenceTransformer
import faiss, numpy as np
embedder = SentenceTransformer("all-MiniLM-L6-v2")
embs = embedder.encode(df['clean_text'].tolist(), convert_to_numpy=True, show_progress_bar=True)
faiss.normalize_L2(embs)
d = embs.shape[1]
index = faiss.IndexFlatIP(d)
index.add(embs)
faiss.write_index(index, "faiss_index.idx")
```

Acknowledgements

I thank Dr. Ahmad Raza Shahid for the assignment, and the open-source community (HuggingFace, SentenceTransformers, FAISS, Gensim) for the models and tools used in this project.