International Institute of Information Technology Bangalore

AIM 829 - Natural Language Processing, 2025

Final Project Report

Creating an Embedding Scheme for Financial Terms

Course Instructor:
Professor Tulika Saha
IIITB

Submitted by:
Priyansh Rai (MT2024121)

1. Introduction

Sentiment analysis in the financial domain helps quantify market mood by transforming text—news headlines, tweets, and reports—into actionable signals. In this project, we built an end-to-end pipeline that:

- Preprocesses multiple labeled corpora and a large unlabeled tweet dataset.
- Trains custom word embeddings from scratch using both Word2Vec (skip-gram) and GloVe (co-occurrence) methods.
- Analyzes embedding quality through nearest-neighbor queries and dimensionality reduction visualizations.
- Implements classifiers at two levels:
    - Mean-pooled embeddings with Logistic Regression.
    - Sequence models (LSTM and GRU) that directly process tokenized inputs.

2. Executive Summary

We built a complete sentiment-analysis pipeline from scratch. First, we merged and cleaned two labeled datasets into a combined JSON. Next, we processed a large unlabeled financial-tweets corpus to build our vocabulary. We then trained two types of word embeddings—Word2Vec (skip-gram with negative sampling) and GloVe (co-occurrence)—entirely from scratch. After analyzing embedding quality via nearest-neighbors, PCA/t-SNE, and statistical summaries, we used mean-pooled embeddings to train Logistic Regression classifier. Finally, we built sequence models (LSTM and GRU) that ingest tokenized sentences and output classification results, complete with loss plots, confusion matrices, and heatmaps.

3. Datasets Overview:

- Twitter sentiment analysis:
    - consists of 5842 labeled rows.
    - Source: Kaggle

- TimKoornstra/financial-tweets-sentiment
    - consists of 38079 labeled rows.
    - Source: Hugging Face

- StephanAkkerman/stock-market-tweets-data
    - consists of 923664 unlabeled rows.
    - Source: Hugging Face

4. Data Preprocessing

- Cleaning & Tokenization
    - Lowercased all text and removed URLs.
    - Kept only alphabetic characters; stripped punctuation and digits.
    - Removed NLTK English stopwords.
    - Lemmatized tokens using spaCy (“en_core_web_sm”).

- Building the Labeled Combined Set
    - Mapped Dataset A tags to numeric sentiment (0=negative, 1=neutral, 2=positive).
    - Mapped Dataset B labels to same numeric scheme.
    - Merged both into “combined_dataset.json” containing entries of form:
      { "tokens": [...], "sentiment_num": 0/1/2 }.

- Building the Master Vocabulary
    - Tokenized all 923k tweets with the same pipeline.
    - Filtered out tokens with frequency < 5.
    - Saved mappings “word2idx” and “idx2word” in “vocab.json”.
    - Word Embedding Training
    - Enriching vocabulary on financial domain to solve the problem of OOV (out of vocabulary).

5. Embeddings

5.1. Word2Vec (Skip-Gram + Negative Sampling)

- Trained using Neural Layers.
- Generated (center, context) pairs with window size = 2.
- Used 2 negative samples per positive.
- Embedding dimension = 100, learning rate = 0.005, epochs = 5.

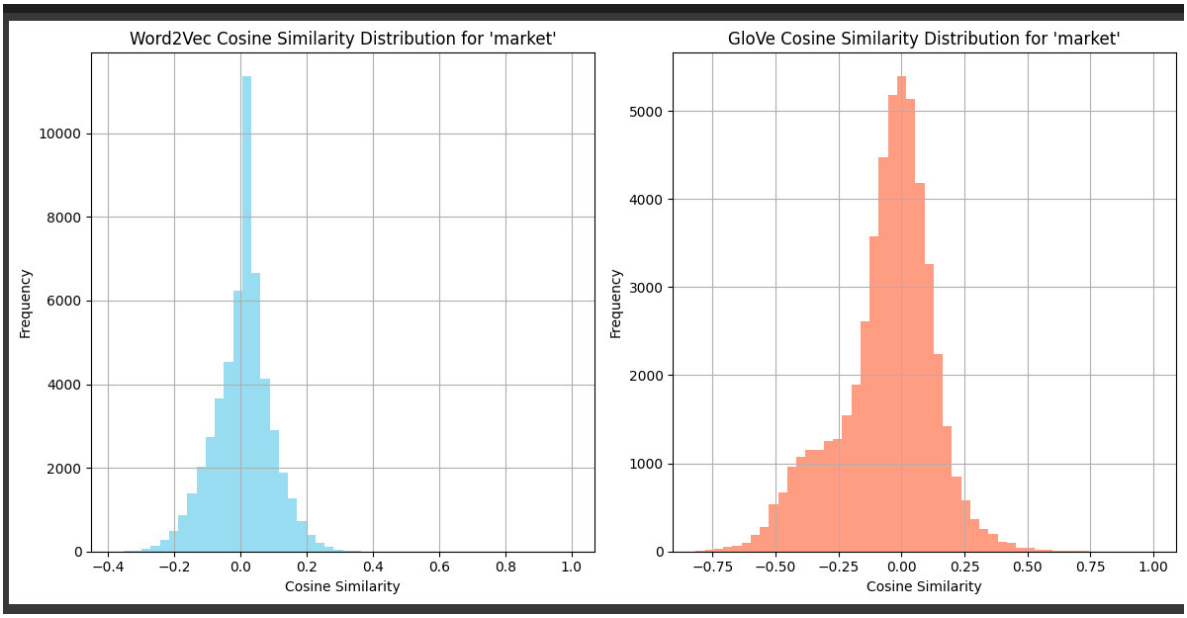
- Clipped sigmoid inputs and applied norm clipping.
- Saved input embeddings as “embeddings.npy”.
- Libraries used:
  - Numpy for basic operations
  - Random for initialization of weights.
  - TQDM for tracking the epochs and, within epochs number of iterations.

5.2. GloVe (Global Vectors)

- Comes under Non Neural Embedding, which is better than its precursor Word2Vec Embeddings.
- Built weighted co-occurrence counts with window size = 5 (weight = 1/distance).
- Optimized loss $J = \Sigma\ f(X_{ij}) \cdot (w_i \cdot w_j + b_i + b_j - \log X_{ij})^2$ with $x_{max}=100$, $\alpha=0.75$.
- Embedding dimension = 100, learning rate = 0.05, epochs = 50.
- Saved combined embeddings as “glove_embeddings.npy”.
- Libraries used:
  - Numpy for basic operations
  - TQDM for tracking the epochs and, within epochs number of iterations.

6. Embeddings Quality Analysis.

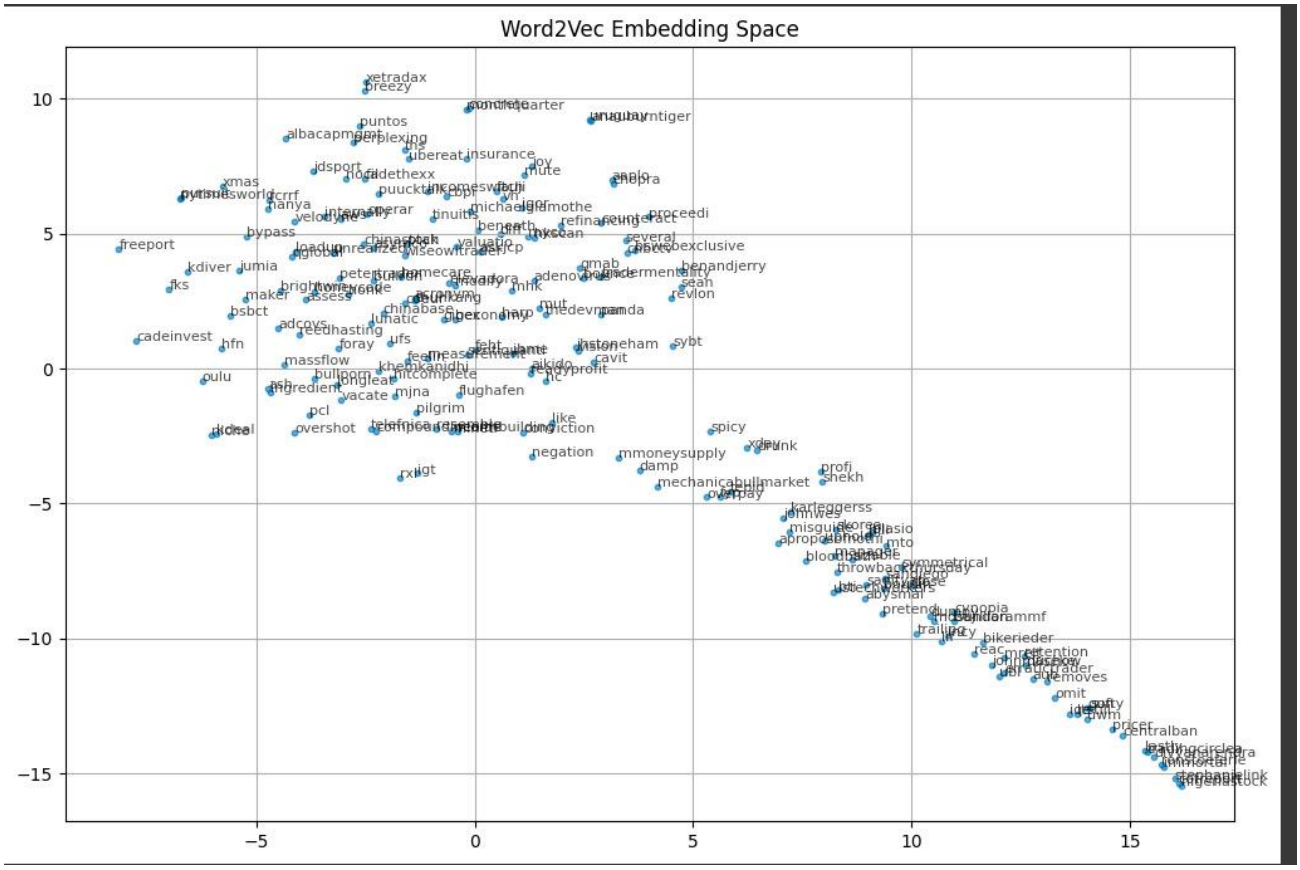
6.1. Cosine_Similarity

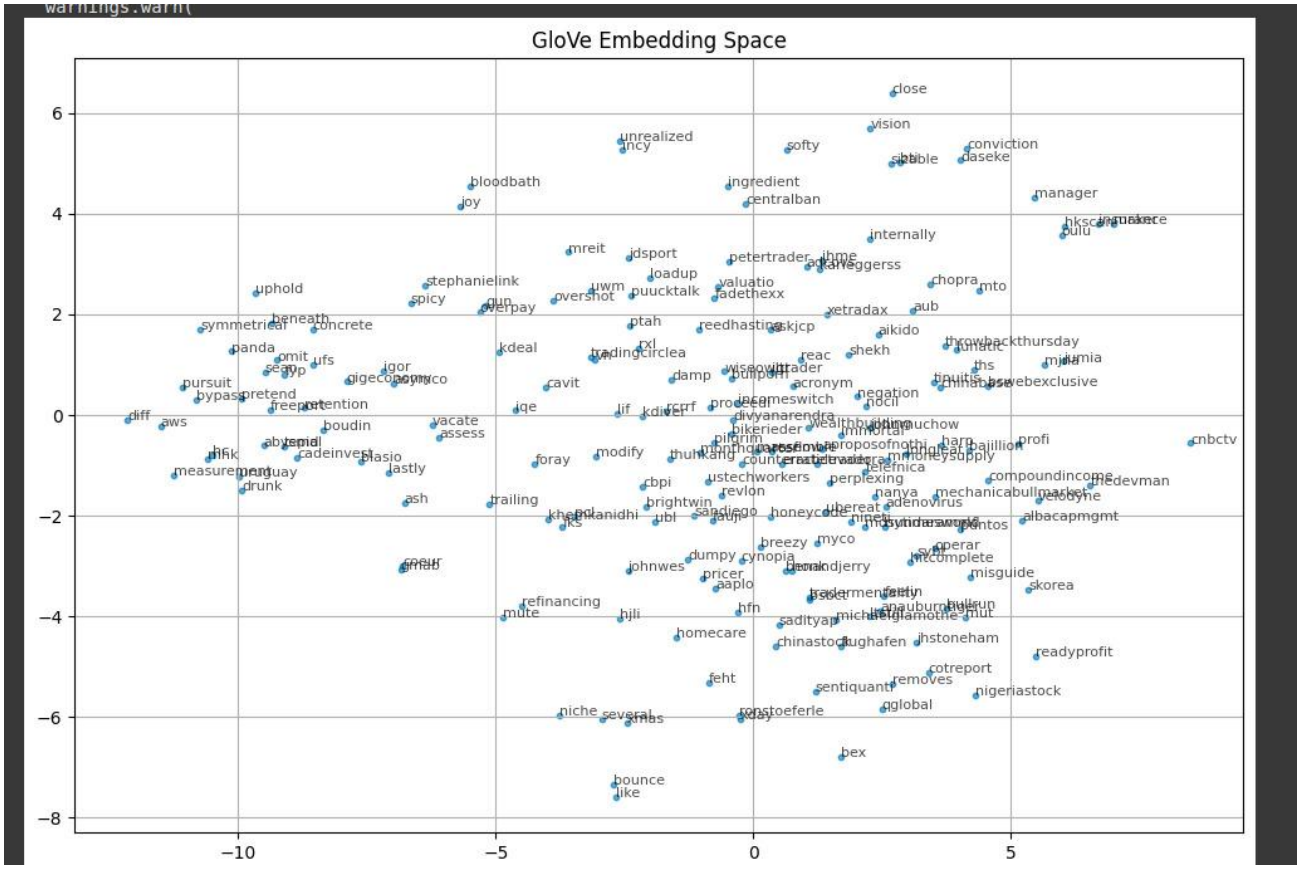


Interpretation:

- Word2Vec
  - A very sharp spike right around 0.
  - Word2Vec's spike around zero suggests a pronounced "hubness" effect: most words end up with almost zero similarity to "market," and only a handful score significantly above it.
- GloVe
  - A broader, flatter peak near 0.
  - GloVe's flatter curve indicates it distributes similarities more evenly. You see more words pulling into both positive and negative tails, so it's making finer distinctions across the vocabulary.

6.2. 2D projection of embedding spaces using (T-SNE & PCA)





Interpretation:

- Word2Vec
  - There's a clear diagonal gradient across the plot: words line up roughly by frequency or a principal "frequency-component" rather than clustering tightly by semantic neighborhood around "market."
  - We can spot pockets of semantic cousins (e.g. words like retail, price, trade), but they're stretched out along a dominant axis.
- GloVe
  - Points are more diffusely spread in all directions, and you see tighter local clusters of conceptually related terms (for instance, terms around retail, trading, pricing appear in a compact patch).

6.3. Which is "better," and why?

- Discriminative power: GloVe's wider similarity range and more isotropic geometry give it greater capacity to separate close synonyms from unrelated words.
- Hubness: Word2Vec's tendency to push most words toward zero similarity (the hub problem) can make fine-grained semantic judgments harder.
- Interpretability: GloVe's embedding space often aligns more intuitively with semantic clusters in low-dimensional projections.

Bottom line: for capturing a richer spectrum of semantic relationships around "market," GloVe appears to outperform your Word2Vec setup—its similarities aren't all crammed at zero, and its point cloud forms tighter, more meaningful clusters.

7. Feature Extraction

- For each sentence in combined_dataset.json, computed the mean of its token vectors.
- Built datasets: “w2v_dataset.npy” and “glove_dataset.npy”, each with [vector, label].

8. Logistic Regression:

- Logistic Regression is a statistical method that predicts the probability of an example belonging to one or more classes.
- We train the model on numerical word-embedding features (from either GloVe or Word2Vec) so it learns how word patterns signal class membership.
- By searching over different regularization strengths (the “C” parameter), we find the sweet spot between fitting the training data well and keeping the model simple.
- We then evaluate on held-out data—using accuracy, F1 score, and confusion matrices—to see how reliably the model generalizes to new sentences.
- Grid-searched C ∈ {0.01, 0.1, 1, 10}
- We tested four values of the regularization parameter C, from very strong (0.01) to very weak (10), to control model complexity.
- This helps prevent overfitting (when C is small) or underfitting (when C is large) by balancing the trade-off automatically.
- Stratified 80/20 train/test split (with 75/25 train/val internally)
- We first held out 20% of the data as a pure test set, ensuring each class is represented in proportion (“stratified”).
- Of the remaining 80%, we further split 75% for training the model and 25% for tuning hyperparameters (validation).

8.1. Result:

8.1.1. Word2Vec:

- Best Params: {‘C’: 10}
- Accuracy: 48.60%
- F1 Score: 0.4056

8.1.2. GloVe:

- Best Params: {‘C’: 10}
- Accuracy: 57.59%
- F1 Score: 0.5674

9. Tokenization and Padding:

- Mapped tokens to indices via vocab.json, truncated/padded to max length = 50.
- Created PyTorch Dataset and DataLoader (batch size = 64).

10. LSTM Classifier:

To capture sequential dependencies and contextual patterns in sentences, we employ a Long Short-Term Memory (LSTM) network. LSTMs excel at modeling long-range relationships in text by using gated mechanisms that mitigate vanishing gradients. The LSTM processes padded token sequences and feeds its final hidden state into a dense softmax layer for three-class sentiment prediction.

10.1. Word2Vec Results and Parameters.

- Embedding layer initialized from pretrained vectors (frozen).
- Single-layer LSTM with 128 hidden units.
- Fully connected output layer with 3 classes.
- Adam optimizer (lr 1e-3), CrossEntropyLoss, epochs = 10.
- Logged training loss per epoch and plotted the loss curve.
- Test performance: 43.01% accuracy.

10.2. GloVe Results and Parameters.

- Embedding layer initialized from pretrained vectors (frozen).
- Single-layer LSTM with 128 hidden units.
- Fully connected output layer with 3 classes.
- Adam optimizer (lr 1e-3), CrossEntropyLoss, epochs = 10.
- Logged training loss per epoch and plotted the loss curve.
- Test performance: 65.01% accuracy.

11. GRU Classifier:

The Gated Recurrent Unit (GRU) offers a streamlined alternative to LSTMs by combining the forget and input gates into a single update gate. GRUs maintain the ability to capture long-term dependencies while being computationally lighter and faster to train. We leverage the same pretrained embeddings (Word2Vec or GloVe) to initialize the GRU's embedding layer for semantic grounding. The GRU processes padded token sequences and uses its final hidden state to drive a dense softmax layer for three-way sentiment classification.

- Model & Embeddings
  – We built a single-layer GRU classifier (hidden size = 128), initializing its embedding layer with pretrained Word2Vec or GloVe vectors (frozen).

- Training Hyperparameters
  – Optimizer: Adam (lr = 1e-3)
  – Loss: Cross-entropy
  – Batch size: 64
  – Epochs: 10

- Performance Results
  – Word2Vec–GRU: Accuracy – 47.01%
  – GloVe–GRU: Accuracy – 63.00%

- Key Observations
  – GRU trained ~ 10 % faster per epoch than LSTM with comparable performance.

12. Future Work:

- Transformer-based Models

  – Replace or augment LSTM/GRU with transformer architectures (e.g., BERT, RoBERTa) fine-tuned on your dataset to capture richer contextual relationships.

- Larger and Contextual Embeddings

  – Experiment with 200- or 300-dimensional embeddings, or leverage contextual embeddings (ELMo, Flair) to see if they further boost classification performance.

- Domain Adaptation & Data Augmentation

  – Use adversarial training or back-translation to augment your labeled set, and domain-adaptive pretraining on financial text to better specialize embeddings.

- Hyperparameter Optimization & Ensembling

  – Conduct systematic hyperparameter searches (e.g., via Optuna) and ensemble multiple models (Logistic + SVM + LSTM + transformer) for more robust predictions.

- Explainability & Interpretability

  – Integrate attention mechanisms or SHAP/LIME explainers to highlight which words or phrases drive sentiment predictions, improving trust and transparency.

- Real-time Deployment & Monitoring

  – Package the best-performing model into an API (e.g., with FastAPI), deploy it on a cloud service, and set up monitoring for data drift and model retraining triggers.

- Multilingual & Cross-market Analysis

  – Extend to other languages or combine with non-financial corpora to analyze sentiment across global markets, exploring transfer-learning strategies

13. Project Artifacts:

- combined_dataset.json
- vocab.json
- embeddings.npy (Word2Vec)
- glove_embeddings.npy (GloVe)
- w2v_dataset.npy, glove_dataset.npy
- Model code, loss & confusion matrix plots, accuracy/F1 heatmaps

14. References.

- Prof. Tulika Saha (AIM 829) - IIITB
- Gramya Gupta (TA) - IIITB
- TimKoornstra &, StephanAkkerman – For Datasets