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

Group Members:
Hrushikesh Nakka (MT2024095)
Naman Samant (MT2024098)
Prabhav Pandey (MT2024115)
Priyansh Rai (MT2024121)

Contents

1 Introduction

Financial Natural Language Processing (NLP) presents unique challenges due to its domain-specific vocabulary, abbreviations, and nuanced expressions. This project aims to develop a custom word embedding scheme tailored to financial text and use it for sentiment analysis on financial tweets. The primary motivation is to improve downstream tasks like classification by creating representations that better capture financial semantics than generic embeddings.

2 Objective and Motivation

Objective: To create a high-quality embedding scheme using domain-specific corpora and evaluate its effectiveness through a downstream sentiment classification task.
Motivation: Standard embeddings trained on general corpora like Wikipedia or Common Crawl do not adequately capture the unique semantics of financial terms. A custom embedding can yield better performance for tasks such as sentiment classification, risk modeling, and market impact assessment.

3 Dataset Description

- **Primary Dataset:** Twitter Financial News Sentiment Dataset
- **Description:** The dataset includes finance-related tweets labeled with sentiment categories:
  - 0 – Bearish
  - 1 – Bullish
  - 2 – Neutral
- **Additional Dataset:** FinancialPhraseBank-v1.0 – A collection of financial texts annotated for sentiment, used optionally for comparison.

4 Workflow

1. Collect and clean financial tweets
2. Preprocess text data: tokenization, lemmatization, stopword removal
3. Train multiple embedding schemes (Word2Vec, FastText, GloVe)
4. Use embeddings in sentiment classification (Logistic Regression, CNN)
5. Evaluate the effectiveness using accuracy, precision, and recall

5 Data Preprocessing

The preprocessing pipeline was implemented in Python using libraries such as NLTK, SpaCy, and regular expressions. It involved cleaning, normalization, and preparation of tweet data for embedding training. The key steps were:

- **Lowercasing:** All tweets were converted to lowercase using Python string operations to ensure uniformity and avoid duplication of tokens due to case differences.
- **Noise Removal:** Tweets often contain URLs, mentions, hashtags, and special characters. These were removed using regular expressions:
  - URLs were stripped using regex pattern `http[s]?://\S+`
  - Twitter handles (e.g., @user) and hashtags (#topic) were removed using appropriate regex
  - Numbers and punctuations were removed using `string.punctuation` and regex
- **Whitespace Normalization:** Multiple spaces were reduced to single spaces to clean up the text structure.
- **Tokenization:** Each tweet was tokenized using SpaCy's English language model `en_core_web_sm`, which provides more accurate tokenization than standard string splitting.
- **Stopword Removal:** Common English stopwords were removed using SpaCy's built-in stopword list, which helped retain only meaningful tokens for embedding training.
- **Lemmatization:** Words were reduced to their base forms using SpaCy's lemmatizer, which helps reduce sparsity in the vocabulary and ensures better semantic grouping of similar words.
- **Short Token Removal:** Tokens with length ≤ 2 were removed, as they mostly represented noise (e.g., "rt", "u", "ok").
- **Final Structure:** The cleaned and tokenized text was stored in a new column named `cleaned_text`, which was later used for training the embedding models.

This preprocessing pipeline ensured the corpus was noise-free, semantically rich, and consistent for downstream embedding training and classification tasks.

6 Data Augmentation Strategy

To enhance the diversity and volume of the dataset for more robust model training, we applied context-aware and rule-based text augmentation techniques to the original tokenized sentences. These augmentations were designed to maintain semantic coherence while introducing lexical diversity.

Techniques Used:

- **Contextual Word Embedding Substitution (BERT-based):** Contextually substitutes words using a masked language model (`bert-base-uncased`). This allows for syntactically and semantically consistent sentence variants with altered vocabulary.
- **WordNet Synonym Replacement:** Replaces non-stopwords with their synonyms derived from the WordNet lexical database, preserving the sentence's core intent while diversifying expression.

Augmentation Rules:

- **Short Sentences** (fewer than 5 words): augmented 3 times to enrich underrepresented short examples.
- **Sentences with Rare Words**: augmented 2 times to reinforce uncommon vocabulary in the dataset.
- **Very Long Sentences** (more than 15 words): skipped to avoid introducing excessive noise or unnatural phrasing.
- **All Other Sentences**: augmented once to provide balanced variation.

Each generated sentence variant preserved the sentiment label of the original and was appended to the dataset, effectively increasing training samples while maintaining class consistency. This augmentation strategy helped improve model generalization and class balance without compromising linguistic integrity.

7 Embedding Scheme Creation

We implemented and compared three word embedding techniques—Word2Vec, FastText, and GloVe—to capture the semantics of financial text. The models were trained on the cleaned tweet corpus created during preprocessing.

7.1 Word2Vec

A Skip-Gram Word2Vec model was trained using Gensim on the financial tweet corpus. The key hyperparameters used were:

- `vector_size = 100`
- `window = 5`
- `min_count = 2`
- `sg = 1` (indicates Skip-Gram architecture)
- `epochs = 10`

We generated embeddings for each word in the vocabulary and stored them in a dictionary for later use. To verify the semantic quality of embeddings, nearest neighbor queries were conducted. For example, querying the word "stock" returned semantically similar terms such as "equity", "market", and "trading".

7.2 FastText

FastText was trained with the same tweet corpus and similar hyperparameters as Word2Vec, but with the key difference of incorporating subword information. Each word is represented as a bag of character n-grams, which helps capture meaning even for rare or unseen words. Parameters used included:

- `vector_size = 100`
- `window = 5`
- `min_count = 2`
- `sg = 1`
- `epochs = 10`

We used pretrained Word2Vec vectors to initialize FastText training, allowing it to converge faster and retain previously learned representations. The FastText model proved effective in generating meaningful embeddings for misspelled terms and rarely used jargon.

7.3 GloVe

Due to resource constraints, we did not train GloVe from scratch. Instead, we imported pretrained GloVe vectors (100-dimensional) trained on 6 billion tokens from the Common Crawl corpus. The pretrained vectors were filtered to retain only the words appearing in our financial corpus.

Each tweet was tokenized, and the GloVe vectors corresponding to known words were averaged to create sentence-level representations. Although GloVe embeddings lacked domain-specific contextuality, they provided valuable global co-occurrence knowledge.

7.4 Combined Vector Representation

To leverage the strengths of all three embedding methods, we created a hybrid embedding by taking a weighted average of Word2Vec, FastText, and GloVe vectors:

$$\mathbf{v}_{\text{hybrid}} = 0.4 \cdot \mathbf{v}_{\text{Word2Vec}} + 0.4 \cdot \mathbf{v}_{\text{FastText}} + 0.2 \cdot \mathbf{v}_{\text{GloVe}}$$

This combination showed improved performance in downstream classification tasks. The hybrid embeddings captured both local context (via Word2Vec and FastText) and global semantics (via GloVe), making them robust across various sentiment cases.

8 Sentiment Classification Models

To evaluate the effectiveness of various embeddings, we employed four modeling approaches: Logistic Regression, Support Vector Machine (SVM), Random Forest, and Convolutional Neural Network (CNN). Each model offers distinct advantages in terms of complexity, interpretability, and ability to capture linear or non-linear relationships.

8.1 Logistic Regression

Logistic Regression is a linear classifier that models the probability of a class using the sigmoid (or softmax for multiclass) function applied to a weighted sum of input features. It is simple, interpretable, and effective as a baseline. It has a low risk of overfitting on high-dimensional text embeddings and allows us to examine which dimensions most influence sentiment. Logistic Regression achieved its best performance with hybrid embeddings, suggesting the feature space was linearly separable.

8.2 Support Vector Machine (SVM)

SVM attempts to find the optimal decision boundary by maximizing the margin between classes. Using the RBF kernel, the SVM captured non-linear relationships in the embedding space. It performed better than Logistic Regression across most embedding types, especially on hybrid and combined vectors, but required more computational time.

8.3 Random Forest

Random Forest is an ensemble of decision trees trained on different subsets of data and features. It models complex non-linear interactions and provides insight into feature importance. It performed well across all embeddings and showed robustness against overfitting. It also improved over Logistic Regression, particularly when trained on enriched or combined embeddings.

8.4 Convolutional Neural Network (CNN)

CNNs applied 1D convolutions over token sequences to learn local n-gram-like features in the embedding space. The model learned both word-level patterns and phrase-level sentiment cues. With global max pooling and dense layers, CNN outperformed all other models—especially when trained with hybrid embeddings. It combined the strengths of both learned embeddings and structural awareness of text.

9 Model Implementation

The models were implemented in Python using TensorFlow/Keras and scikit-learn:

- **Logistic Regression:** A dense output layer with softmax activation, trained on averaged tweet embeddings.
- **SVM and Random Forest:** Implemented using scikit-learn, trained on sentence-level embeddings.
- **CNN:** Used a frozen embedding layer followed by Conv1D, ReLU, MaxPooling, and Dense layers.

10 Evaluation

We evaluated model performance using Accuracy and F1-score across four models (Logistic Regression, SVM, Random Forest, and CNN) on each of the four embedding types. Classification reports were also analyzed for more granular insights into class-wise behavior.

Metrics Used

- **Accuracy:** The percentage of correctly predicted labels out of total predictions.
- **F1-score:** Harmonic mean of precision and recall, weighted across classes.

1. Word2Vec Embeddings

Model	**Accuracy**	**F1-score (weighted)**
Logistic Regression	**0.4548**	**0.3763**
SVM	0.3944	–
Random Forest	0.3944	–
CNN	0.4181	–

Table 1: Performance on Final Word2Vec Embeddings

Best Model: Logistic Regression (Accuracy: 0.4548)

2. Hybrid Embeddings (Word2Vec + FastText)

Model	**Accuracy**	**F1-score (weighted)**
Logistic Regression	**0.5196**	**0.4749**
SVM	0.4283	0.2803
Random Forest	0.4775	–
CNN	0.4709	–

Table 2: Performance on Hybrid Embeddings

Best Model: Logistic Regression (Accuracy: 0.5196)

3. Combined GloVe + Word2Vec + FastText Embeddings

Model	Accuracy	F1-score (weighted)
Logistic Regression	**0.5083**	0.4638
SVM	0.4423	0.3109
Random Forest	0.4766	–
CNN	0.4821	–

Table 3: Performance on Combined Embeddings

Best Model: Logistic Regression (Accuracy: 0.5083)

4. GloVe + Word2Vec Enhanced Embeddings

Model	Accuracy	F1-score (weighted)
Logistic Regression	**0.4619**	**0.3883**
SVM	0.4486	0.3290
Random Forest	0.4028	–
CNN	0.4487	–

Table 4: Performance on GloVe + W2V Enhanced Embeddings

Best Model: Logistic Regression (Accuracy: 0.4619)

Key Observations

- **Logistic Regression** surprisingly outperformed other models across all embedding types in terms of both accuracy and F1-score.
- **CNN and Random Forest** showed competitive accuracy but lacked consistency in F1-score due to potential class imbalance.
- **SVM** underperformed on most embeddings, particularly for the minority class (-1), often with precision and recall near zero.
- The **hybrid embedding** paired with Logistic Regression yielded the best overall performance across the board.

11 Future Work

While this project successfully demonstrated the creation and evaluation of custom embedding schemes for financial sentiment classification, there are several avenues for future exploration:

- **Contextual Embeddings:** Incorporate transformer-based models such as BERT, RoBERTa, or FinBERT to capture contextual dependencies in financial language. These models can dynamically adjust word meanings based on surrounding words.
- **Multi-task Learning:** Extend the current model to perform related tasks such as named entity recognition (NER) or event extraction jointly with sentiment analysis to improve generalization.
- **Time-aware Modeling:** Introduce temporal components to account for how sentiment trends evolve over time in financial data, potentially improving predictive performance.
- **Real-time Deployment:** Convert the trained models into APIs or lightweight services suitable for integration with real-time social media monitoring tools or trading platforms.
- **Explainability:** Integrate model explainability techniques (like SHAP or LIME) to better understand which financial terms or patterns influence sentiment predictions, especially for high-stakes decision-making.
- **Cross-Domain Transfer:** Evaluate how well the custom embeddings and trained models transfer to other financial datasets, such as news headlines, earnings reports, or analyst summaries.

12 Conclusion

This project successfully demonstrated the creation of a domain-specific embedding scheme tailored for financial sentiment analysis. By training custom Word2Vec and FastText models on finance-related tweets and leveraging CNN for classification, we achieved promising results. Future work includes incorporating contextual embeddings like BERT and extending analysis to other downstream financial tasks like fraud detection or volatility prediction.

13 Work Distribution

The project was a collaborative effort, with each team member contributing to specific components of the workflow. The responsibilities were distributed as follows:

- **Hrushikesh Nakka (MT2024095)**
  Data Preprocessing and Augmentation
  Responsible for cleaning, tokenizing, and lemmatizing the financial tweets. Implemented text augmentation techniques using contextual and lexical approaches to enhance dataset diversity and balance.

- **Naman Samant (MT2024098)**
  Embedding Training and Vector Engineering
  Led the design and implementation of custom word embedding schemes including Word2Vec, FastText, and hybrid embeddings. Tuned embedding hyperparameters and managed integration across models.

- **Prabhav Pandey (MT2024115)**
  Embedding Research and Model Development
  Conducted in-depth research on embedding strategies, analyzed vector spaces, and contributed to the development of machine learning and deep learning models for sentiment classification.

- **Priyansh Rai (MT2024121)**
  Embedding Research and Model Development
  Worked on evaluating embedding effectiveness across models, supported model implementation (SVM, CNN, Logistic Regression), and contributed to classification performance analysis.