NLP Sentiment Analysis of Financial News

Laurenz Gilbert, Ariana Sahitaj
Information Systems Management, TU Berlin
Matr.Nr.: 0508033, 0504729
{gilbert,ariana.sahitaj}@campus.tu-berlin.de

Advised by: Dr. Salar Mohtaj

ABSTRACT

In this report, we address the task of sentiment analysis on financial news using natural language processing techniques. Our approach follows four main steps: data exploration, text pre-processing, sentiment classification, and semantic similarity analysis. We begin by extracting statistical insights and visual summaries from a labeled dataset of financial sentences to understand the distribution of sentiment labels and key vocabulary patterns. Various pre-processing strategies are then applied to normalize and clean the text. For classification, we compare traditional machine learning models, such as Naive Bayes, with a feedforward neural network, evaluating the effect of different vectorization methods and pre-processing pipelines. In addition, we analyze word similarities using a pointwise mutual information (PMI) approach and experiment with a hybrid model that combines transformer-based embeddings (XLM-RoBERTa) and a recurrent neural network. The report concludes with a comparative performance analysis across models and insights into pre-processing impact, offering a comprehensive view of financial sentiment detection using NLP.

Reference Format:
Laurenz Gilbert, Ariana Sahitaj. 2020. NLP Sentiment Analysis of Financial News. In *NYUAD Capstone Seminar Reports, Spring 2020, Abu Dhabi, UAE.* 7 pages.

INTRODUCTION

Sentiment analysis, also known as opinion mining, is a tool within natural language processing (NLP) to assess public perception and market reactions based on textual information [2, 6]. Traditionally, market sentiment about a company or a sector has been measured through confidence indicators derived from regular surveys, which often become quickly outdated and fail to capture real-time market dynamics [5]. Given the fast-paced nature of financial markets, there is a need for more agile methods capable of capturing sentiments effectively and in real-time [3, 4]. Recent advancements in NLP techniques have demonstrated significant improvements in automatically classifying sentiments from textual data, notably through machine learning and deep learning approaches [1]. This project applies NLP techniques to classify financial news into positive, neutral, and negative categories using the `Sentences_50Agree` dataset. We compare traditional methods like Naïve Bayes and deep learning models such as feed-forward neural networks (Section 3), evaluating various pre-processing and vectorization strategies. We further assess semantic similarity via pointwise mutual information (PMI) (Section 4) and, as a bonus, test a hybrid model combining XLM-RoBERTa embeddings with a recurrent neural network (Section 4.1).

1 EXTRACT INSIGHTS FROM DATA

To gain a comprehensive understanding of the dataset and its structure, we conducted an exploratory analysis focused on sentence distribution, vocabulary statistics, and word usage across sentiment classes. The dataset consists of 4,846 labeled sentences, each classified into one of three categories: neutral, positive, or negative. In the first step, we analyzed the distribution of sentiment labels (see Figure 1, left). The dataset is not perfectly balanced: the majority of sentences are labeled as neutral (2,879), followed by positive (1,363), and negative (604). This class imbalance is important to keep in mind when training models, as it could influence prediction outcomes. Next, we examined the length of sentences in terms of character count (Figure 1, center). On average, positive sentences are slightly longer than negative and neutral ones. The mean sentence lengths are 136 characters (positive), 126 (negative), and 125 (neutral), while the medians are 128, 118, and 116 respectively. The distribution is visualized with violin plots, which also reveal that while sentence lengths vary, there is a consistent range across all three sentiment types. Figure 1 (right) shows the number of unique words per sentiment. The neutral class has the highest vocabulary diversity with 6,794 unique words, followed by positive with 3,893, and negative with 1,825. Across the entire dataset,

Capstone Seminar, Spring 2020, Abu Dhabi, UAE

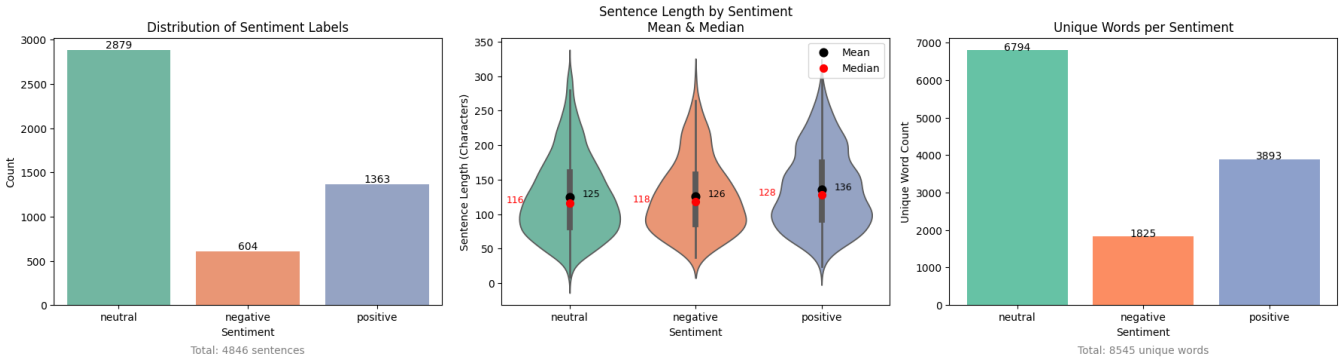

Figure 1: Sentiment Analysis of Labeled Sentences

we identified 8,545 unique words, indicating some overlap between categories but also a notable difference in lexical richness among them. In the second stage of analysis, we explored word frequency patterns. Figure 2 presents the 15

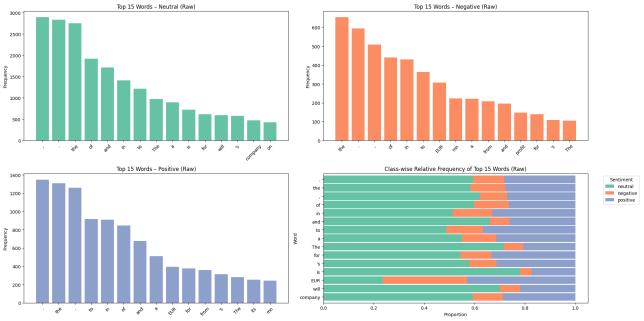

Figure 2: Most Frequent Words Across Sentiments

most frequent words per sentiment class using raw (unprocessed) text. As expected, the top words are dominated by

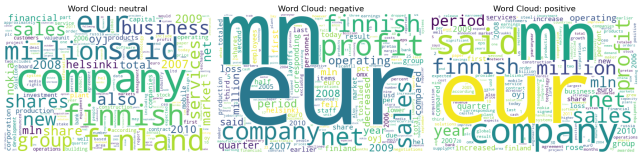

Figure 3: Word clouds for each sentiment category, showing most frequent words pre-processing. Word size represents relative frequency within each class.

punctuation and common stopwords like "the", "and", "of", or ".". These are high-frequency but low-information terms, providing little insight into sentiment-specific vocabulary. This underscores the importance of pre-processing to extract sentiment-relevant linguistic patterns.

This is also seen in the generated word clouds for each sentiment category (Figure 3). For instance, the positive word

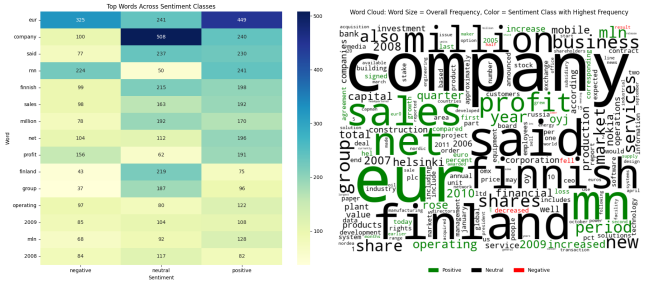

Figure 4: Comparison of the 15 most frequent words across sentiment classes. The heatmap (left) shows absolute word frequencies per class. The word cloud (right) displays overall word frequency as size, with color indicating the sentiment class in which the word appears most frequently.

cloud highlights terms such as eur, mn, company which is also very similar to the negative word cloud. To make it more precise, we pre-processed to surface sentiment-relevant patterns in the data to better highlight meaningful patterns. That included lowercasing, punctuation removal, and stopword filtering. After pre-processing, the **median word count** was 11 for both neutral and positive sentences, and 10 for negative ones. The **interquartile range** (IQR) was 7–15 words for neutral, 8–14 for negative, and 8–16 for positive. The **maximum observed word counts** (excluding outliers) were 36, 29, and 31 words for neutral, negative, and positive respectively, while the minimum was 0 for neutral and 2 for both negative and positive. In the final step, we analyzed the distribution of top words across sentiments more quantitatively. Figure 4 (left) presents a heatmap showing the frequency of the 15 most common words across all sentiment classes after pre-processing. For example, the word eur appears frequently in all classes but is most common in positive sentences. Similarly, company and said are dominant in neutral texts. We also created a color-coded word cloud

Pipeline	Lower	Currency	Unicode	Possessive	Percent	Number Abbr.	Number Range	Hyphens	Commas	Punct.	Lem. / Stem / Stopw.
(a) standard	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓ / ✗ / ✗
(b) with_stopword_removal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓ / ✗ / ✓
(c) basic_cleaning	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗ / ✗ / ✗
(d) stemming_only	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗ / ✓ / ✗
(e) aggressive_normalization	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓ / ✓ / ✓
(f) no_number_currency_replacement	✓	✗	✓	✓	✗	✗	✓	✓	✓	✓	✓ / ✗ / ✗
(g) tokenize_and_lowercase_only	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗ / ✗ / ✗

Table 1: Overview of Pre-processing Pipelines: Enabled (✓) and Disabled (✗) Options for Text Normalization, Token Filtering, and Morphological Processing.

(Figure 4, right), where word size indicates overall frequency, and color represents the sentiment class in which the word appears most frequently (green: positive, red: negative, black: neutral). This visualization helps quickly identify both sentiment alignment and term salience. Words such as profit, increase, and net are green (positive); fell, decreased, and loss appear in red (negative); while terms like company, sales, and eur are black (neutral), reflecting their more balanced or context-dependent use. The analysis shows that the dataset has a moderate class imbalance, with neutral sentences being the most frequent and lexically diverse.

2 PRE-PROCESSING

In order to prepare the financial news texts for sentiment classification, we applied a series of pre-processing steps, as seen in Table 1, aimed at reducing lexical variation, normalizing numeric and symbolic expressions, and facilitating semantic interpretation. The pipeline of 7 modi included lowercasing all text to ensure case-insensitive matching. Currency symbols (such as $, €, £) and percent signs (%) were replaced with their word equivalents (*dollar*, *euro*, *pound*, *percent*) to enhance interpretability by both statistical and neural models. Numerical abbreviations (e.g., "5m" to "5 million") were standardized, and numeric ranges (e.g., "5-10") were rewritten as explicit spans ("5 to 10") for consistency. We removed possessive markers ('s), commas in numbers, hyphens within words, and most punctuation to reduce noise and token fragmentation. Whitespace was normalized to avoid tokenization artifacts. All pre-processing steps were implemented modularly in code using boolean flags, allowing flexible combinations of cleaning operations. For semantic normalization, we applied lemmatization using POS-aware rules, reducing words to their canonical forms while preserving grammatical roles. Stemming was tested in some configurations but omitted from the main pipeline due to its tendency to distort word meaning. Similarly, stopword removal was evaluated in alternative setups but excluded from the default configuration to preserve potentially meaningful syntactic and contextual signals (especially in short financial snippets where function words may carry sentiment-relevant information).

3 TEXT CLASSIFICATION

This section analyzes sentiment classification in financial news using two models: a Naive Bayes classifier and a Feed-Forward Neural Network (FFNN). We systematically compare multiple pre-processing pipelines and three vectorization strategies, perform detailed error analysis, and evaluate both multi-class and binary setups. The aim is to assess how feature engineering and model complexity affect performance, using standard evaluation metrics. The dataset was split into 80% training and 20% test data. To address class imbalance, we applied RandomOverSampler from the imblearn library, ensuring fair conditions for both models.

3.1 Naive Bayes

We evaluated six vectorization methods: CountVectorizer with unigrams and bigrams, TfidfVectorizer with unigrams and bigrams, and HashingVectorizer with unigrams and bigrams. Each was tested across all seven preprocessing settings (a) through (g). In addition to the multi-class setup, we trained a binary Naive Bayes classifier by excluding the neutral class and focusing only on positive and negative instances. The best performance was achieved with pipeline (f), which avoids replacing numbers or currency symbols and does not apply stopword removal or lemmatization/stemming. Combined with CountVectorizer (bigrams), this configuration reached 0.69 accuracy and 0.70 weighted F1. The worst results were seen with pipelines (e) and (b) when using TfidfVectorizer with bigrams, each reaching only 0.52 accuracy and 0.55 weighted F1. These configurations involve either aggressive normalization or isolated stopword removal, which may strip away relevant information needed by Naive Bayes to discriminate between classes. Table 3 presents the performance of a binary Naive Bayes classifier (excluding neutral instances) using CountVectorizer across all seven preprocessing pipelines (a) to (g). All pipelines achieve comparable results, with accuracy values ranging from 0.77 to 0.79 and weighted F1 scores between 0.77 and 0.80. The best-performing pipelines are (b), (d), and (f), each reaching 0.79 accuracy and 0.80 weighted F1. Even the lowest-performing setup, pipeline (c), still achieves a solid 0.77 accuracy, indicating that binary classification is

Pipeline	Vectorizer	Accuracy	Macro F1	Weighted F1
Naive Bayes				
(a)	CountVectorizer_Unigram	0.67	0.62	0.68
(a)	CountVectorizer_Bigram	0.68	0.64	0.69
(a)	TfidfVectorizer_Unigram	0.66	0.62	0.67
(a)	TfidfVectorizer_Bigram	0.65	0.60	0.66
(a)	HashingVectorizer_Unigram	0.65	0.61	0.66
(a)	HashingVectorizer_Bigram	0.65	0.61	0.67
(b)	CountVectorizer_Unigram	0.65	0.60	0.66
(b)	CountVectorizer_Bigram	0.55	0.52	0.57
(b)	TfidfVectorizer_Unigram	0.65	0.61	0.66
(b)	TfidfVectorizer_Bigram	0.52	0.50	0.55
(b)	HashingVectorizer_Unigram	0.63	0.59	0.64
(b)	HashingVectorizer_Bigram	0.54	0.50	0.56
(c)	CountVectorizer_Unigram	0.68	0.63	0.68
(c)	CountVectorizer_Bigram	0.68	0.63	0.69
(c)	TfidfVectorizer_Unigram	0.66	0.62	0.67
(c)	TfidfVectorizer_Bigram	0.65	0.61	0.66
(c)	HashingVectorizer_Unigram	0.65	0.60	0.66
(c)	HashingVectorizer_Bigram	0.64	0.59	0.65
(d)	CountVectorizer_Unigram	0.68	0.63	0.68
(d)	CountVectorizer_Bigram	0.68	0.63	0.69
(d)	TfidfVectorizer_Unigram	0.68	0.63	0.68
(d)	TfidfVectorizer_Bigram	0.65	0.61	0.67
(d)	HashingVectorizer_Unigram	0.65	0.61	0.66
(d)	HashingVectorizer_Bigram	0.63	0.58	0.64
(e)	CountVectorizer_Unigram	0.66	0.61	0.67
(e)	CountVectorizer_Bigram	0.55	0.52	0.58
(e)	TfidfVectorizer_Unigram	0.65	0.61	0.66
(e)	TfidfVectorizer_Bigram	0.52	0.50	0.55
(e)	HashingVectorizer_Unigram	0.62	0.58	0.63
(e)	HashingVectorizer_Bigram	0.52	0.49	0.54
(f)	CountVectorizer_Unigram	0.67	0.62	0.68
(f)	CountVectorizer_Bigram	0.69	0.64	0.70
(f)	TfidfVectorizer_Unigram	0.66	0.62	0.67
(f)	TfidfVectorizer_Bigram	0.66	0.61	0.67
(f)	HashingVectorizer_Unigram	0.65	0.61	0.67
(f)	HashingVectorizer_Bigram	0.66	0.62	0.67
(g)	CountVectorizer_Unigram	0.68	0.63	0.68
(g)	CountVectorizer_Bigram	0.69	0.64	0.70
(g)	TfidfVectorizer_Unigram	0.67	0.62	0.67
(g)	TfidfVectorizer_Bigram	0.66	0.61	0.67
(g)	HashingVectorizer_Unigram	0.66	0.61	0.67
(g)	HashingVectorizer_Bigram	0.64	0.59	0.66

Table 2: Naive Bayes across Different Text Preprocessing Pipelines and Vectorization Methods.

more robust across preprocessing variations than the multiclass setting. Further results using other vectorizers (Tfidf and Hashing, with both unigrams and bigrams) were evaluated as well but are omitted here to keep the table concise. They are available in the code.

Pipeline	Accuracy	Macro F1	Weighted F1
Binary Naive Bayes			
(a)	0.78	0.77	0.79
(b)	0.79	0.77	0.80
(c)	0.77	0.75	0.77
(d)	0.79	0.77	0.80
(e)	0.78	0.76	0.78
(f)	0.79	0.77	0.80
(g)	0.79	0.77	0.79

Table 3: Binary Naive Bayes (pos/neg) across Different Text Pre-processing Pipelines and representative for the CountVectorizer.

3.1.1 Error Analysis and Limitations. A Naive Bayes classifier assumes word independence and uses bag-of-words logic. It lacks awareness of grammar, syntax, or semantic structure. This limitation causes several recurring error types as seen in Table 4 at top. The Naive Bayes classifier relies on the bag-of-words assumption and treats words in isolation, without understanding grammar, word order, or context. This limitation leads to several recurring errors. First, it frequently misinterprets context-sensitive terms. Words like "high" or "profit" are misclassified due to previously learned associations, regardless of their actual meaning in context. Second, the model struggles to distinguish neutral statements from genuine sentiment, often misclassifying factual content due to sentiment-laden keywords like "loan" or "support." Third, it fails with comparative statements, it cannot assess whether a change (e.g., reduced losses) is positive or negative. Lastly, company names, codes, and financial jargon introduce noise, weaken meaningful signals and reducing classification accuracy. These examples illustrate the model's core limitation: a lack of semantic and contextual understanding.

3.2 Feed-Forward Neural Network

The feedforward neural network performs best with pipeline (e), which applies aggressive normalization including lemmatization, stemming, and stopword removal, as seen in Table 5. The neural network consists of a fully connected architecture with two hidden layers. The input size corresponds to the number of features from the vectorization step. The first dense layer has 128 neurons with ReLU activation, followed by a dropout layer (rate 0.3). The second dense layer has 64 neurons, also with ReLU and dropout. The output layer uses softmax activation and has either two units (binary classification) or one per class (multi-class setup). The model is trained using the Adam optimizer and categorical cross-entropy loss, with accuracy as the main metric. Training runs for 10 epochs with a batch size of 32, using 10% of the training data for validation. To address class imbalance, RandomOverSampler is applied before training. Labels are encoded with LabelEncoder and converted to one-hot vectors to match the softmax output format. This setup consistently achieves the highest accuracy and F1 scores (0.78) across all vectorizers. In contrast, pipeline (b), which removes stopwords without stemming, performs worst, especially with bigrams (accuracy: 0.68, macro F1: 0.56). Lemmatization alone (pipeline a) improves performance but does not outperform the more comprehensive pipeline (e). Stemming (pipeline d) does not reduce performance and even contributes positively when combined with other steps. Stopword removal alone (pipeline b) leads to worse performance, suggesting that function words carry useful sentiment signals in financial text. Across all pipelines and vectorizers, the feedforward neural network (FFNN) consistently outperforms Naive Bayes in sentiment classification. While Naive Bayes reaches a maximum accuracy of 0.69 (pipelines f/g with CountVectorizer

Text Snippet	True Label	Predicted	Error Cause
Naive Bayes			
"the diesel margin have remain high"	Positive	Negative	Misinterpreted "high" as negative, ignoring context of "margin"
"operating profit total eur 5.5 million up from eur 0.7 million"	Positive	Negative	Financial jargon likely misclassified from learned negative associations
"the ore body be sufficient to support anticipated production for at least 46 year"	Neutral	Positive	Words like "support" treated as sentiment-bearing
"...have a total of eur 23.0 million in loan..."	Neutral	Negative	"Loan" interpreted as inherently negative
"pretax loss... eur 0.3 million compare to a loss of eur 2.2 million"	Positive	Negative	Model fails to grasp improvement in comparative structure
"sale of ragutis... decline by 11.2 percent..."	Negative	Positive	"Decline" missed, overwhelmed by neutral terms
"nordic aluminium plc omx helsinki noa1v report..."	Neutral	Negative	Noisy input: named entities dilute signal words
Feedforward Neural Network			
"aspocomp intend to set up a plant to manufacture print circuit board with an investment of rs310 crore"	Neutral	Positive	Neutral planning statement misinterpreted as positive due to investment-related keywords
"the diesel margin have remain high"	Positive	Negative	Domain-specific polarity misunderstood: "high margin" is positive, but "high" treated generically
"helsinki afx outokumpu say it have agree to sell the hitura nickel mine..."	Neutral	Negative	Location and entity names introduce noise; action wrongly associated with negative sentiment
"employer-employee talk concern about 500 people"	Neutral	Negative	Keywords like "concern" matched with negative sentiment without understanding the neutral context of labor talks
"sale decline 11.2 percent year-on-year"	Negative	Positive	Decline underweighted; positive terms like "sale" dominate sparse feature representation
"pretax loss of eur 0.3 million compare to a loss of eur 2.2 million"	Positive	Negative	Improvement through comparison missed; lacks contextual reasoning about decreased loss
"profit before tax was eur 3.1 million, compared to a profit of eur 2.0 million"	Positive	Neutral	Positive trend understated due to numerical closeness; lacks magnitude sensitivity

Table 4: Illustrative misclassifications by Naive Bayes and Feedforward Neural Network, categorized by error type.

bigrams), FFNN achieves up to 0.78 (pipeline e across all vectorizers). This performance gap also holds for macro and weighted F1 scores, where FFNN shows up to 10–15 percentage points improvement in challenging setups like pipeline (b) or (e). The results suggest that FFNNs can better capture complex patterns and contextual interactions than the linear assumptions made by Naive Bayes. However, both models are similarly affected by preprocessing: aggressive stopword removal (pipeline b) harms performance in both, while pipeline (e), combining multiple normalization steps, works best for FFNN but poorly for Naive Bayes, indicating model-specific sensitivities.

Table 6 reports binary classification results (positive vs. negative) for the feedforward neural network using CountVectorizer across all preprocessing pipelines. Overall performance is strong and consistent, with accuracy values ranging from 0.83 to 0.86 and macro F1 scores between 0.80 and 0.83. Pipelines (a), (d), (f), and (g) yield the best results with 0.86 accuracy and 0.85–0.86 weighted F1. Even the weakest configuration, pipeline (b), achieves 0.83 accuracy, confirming that FFNN is robust to preprocessing variation in the binary setting.

3.2.1 Comparison: Naive Bayes vs. Feed-Forward Neural Network. Comparing Tables 3 and 6, the feedforward neural network clearly outperforms Naive Bayes in the binary setting. FFNN achieves up to 0.86 accuracy and 0.83 macro F1, whereas Naive Bayes peaks at 0.79 accuracy and 0.77 macro F1. This margin holds across all pipelines, indicating that FFNN can better capture relevant patterns, even with simple bag-of-words input. While both models benefit from well-designed preprocessing, FFNN shows higher

Pipeline	Vectorizer	Accuracy	Macro F1	Weighted F1
Feedforward Neural Network				
(a)	CountVectorizer_Unigram	0.76	0.72	0.76
(a)	CountVectorizer_Bigram	0.74	0.66	0.73
(a)	TfidfVectorizer_Unigram	0.74	0.69	0.74
(a)	TfidfVectorizer_Bigram	0.73	0.65	0.72
(a)	HashingVectorizer_Unigram	0.75	0.71	0.75
(a)	HashingVectorizer_Bigram	0.74	0.66	0.73
(b)	CountVectorizer_Unigram	0.71	0.66	0.71
(b)	CountVectorizer_Bigram	0.71	0.59	0.68
(b)	TfidfVectorizer_Unigram	0.71	0.65	0.71
(b)	TfidfVectorizer_Bigram	0.68	0.58	0.67
(b)	HashingVectorizer_Unigram	0.71	0.66	0.71
(b)	HashingVectorizer_Bigram	0.68	0.56	0.65
(c)	CountVectorizer_Unigram	0.75	0.69	0.75
(c)	CountVectorizer_Bigram	0.73	0.64	0.72
(c)	TfidfVectorizer_Unigram	0.75	0.70	0.75
(c)	TfidfVectorizer_Bigram	0.74	0.65	0.72
(c)	HashingVectorizer_Unigram	0.75	0.71	0.75
(c)	HashingVectorizer_Bigram	0.73	0.64	0.71
(d)	CountVectorizer_Unigram	0.76	0.72	0.75
(d)	CountVectorizer_Bigram	0.74	0.66	0.72
(d)	TfidfVectorizer_Unigram	0.74	0.70	0.74
(d)	TfidfVectorizer_Bigram	0.74	0.65	0.72
(d)	HashingVectorizer_Unigram	0.74	0.71	0.75
(d)	HashingVectorizer_Bigram	0.73	0.64	0.71
(e)	CountVectorizer_Unigram	0.78	0.76	0.78
(e)	CountVectorizer_Bigram	0.78	0.76	0.78
(e)	TfidfVectorizer_Unigram	0.78	0.76	0.78
(e)	TfidfVectorizer_Bigram	0.78	0.76	0.78
(e)	HashingVectorizer_Unigram	0.78	0.76	0.78
(e)	HashingVectorizer_Bigram	0.78	0.76	0.78
(f)	CountVectorizer_Unigram	0.75	0.70	0.75
(f)	CountVectorizer_Bigram	0.75	0.67	0.73
(f)	TfidfVectorizer_Unigram	0.73	0.69	0.73
(f)	TfidfVectorizer_Bigram	0.74	0.66	0.73
(f)	HashingVectorizer_Unigram	0.73	0.68	0.73
(f)	HashingVectorizer_Bigram	0.74	0.65	0.72
(g)	CountVectorizer_Unigram	0.75	0.70	0.74
(g)	CountVectorizer_Bigram	0.74	0.65	0.72
(g)	TfidfVectorizer_Unigram	0.74	0.69	0.74
(g)	TfidfVectorizer_Bigram	0.73	0.67	0.72
(g)	HashingVectorizer_Unigram	0.73	0.68	0.73
(g)	HashingVectorizer_Bigram	0.73	0.66	0.72

Table 5: FFNN across Different Text Pre-processing Pipelines and Vectorization Methods.

stability and stronger gains, particularly when multiple normalization steps are applied. To conclude, across both the multi-class and binary classification settings, the feedforward neural network consistently outperforms the Naive Bayes model. While Naive Bayes benefits from simpler computation and performs reasonably well with unigram-based inputs, it struggles with context-dependent sentiment cues and complex expressions. In contrast, FFNNs are better able to leverage richer feature interactions, leading to more accurate and stable results across a wide range of preprocessing pipelines and vectorizers. This holds true especially in the binary setting, where both models show improved performance compared to the multi-class task, but FFNN maintains a clear advantage throughout.

3.2.2 Error Analysis and Limitations. We also conducted an error analysis for the feedforward neural network model. As shown in Table 4 (bottom), common errors include confusion between neutral and positive statements, incorrect polarity flips (e.g., positive misclassified as negative), and misinterpretation of neutral statements as negative due to keywords like concern or sell. These errors stem from the model's reliance on bag-of-words features, which lack contextual understanding, word order, and semantic nuance. As a result, the model fails to capture domain-specific meanings (e.g., high margin as positive) and struggles with comparative structures or subtle cues. Overall, the feedforward network performs pattern matching over isolated words, which limits its effectiveness for nuanced financial sentiment analysis.

Pipeline	Accuracy	Macro F1	Weighted F1
Binary Feed-Forward Neural Network			
(a)	0.86	0.83	0.85
(b)	0.83	0.80	0.83
(c)	0.85	0.83	0.85
(d)	0.86	0.83	0.85
(e)	0.84	0.81	0.83
(f)	0.86	0.83	0.86
(g)	0.86	0.82	0.85

Table 6: Binary FFNN (pos/neg) across Different Text Pre-processing Pipelines and representative for the CountVectorizer.

4 PMI-BASED WORD SIMILARITY

This task involved manually implementing **Pointwise Mutual Information (PMI)** to measure semantic similarity between words based on their co-occurrence frequencies within a window size of 1. PMI quantifies the association strength between word pairs without relying on pre-built library functions. To compute PMI scores, the dataset was tokenized and preprocessed using a comprehensive pipeline that included lowercasing, punctuation and stopword removal, and lemmatization. The resulting tokens were flattened into a single sequence to extract adjacent word pairs. To avoid duplicates, all pairs were stored in sorted order (e.g., `("bank", "issue")`). Given the large vocabulary and skewed frequency distribution, the resulting co-occurrence matrix was highly sparse. To reduce memory overhead and maintain interpretability, we avoided generating highdimensional sparse embeddings and instead used direct PMI scores to capture local semantic associations. The PMI formula used was $\text{PMI}(w_1, w_2) = \log_2 \left(\frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)} \right)$ where P(w_1, w_2) is the empirical joint probability of the word pair and P(w_1) and P(w_2) are the marginal probabilities based on word counts. After calculating the PMI matrix, we randomly sampled 10 meaningful words (length > 3, excluding function words) from the vocabulary and extracted the top 3-5 associated terms per word, ranked by PMI value.

4.1 Results

The following table 7 presents the 10 randomly selected words and their top PMI-based associations. The PMI-based associations reveal interpretable and domain-relevant semantic relationships. For example, the strong association between `goldman` and `sachs` reflects the well-known financial entity *Goldman Sachs*, frequently mentioned in financial news. Similarly, `employee` is linked to `whitecollar`, `society`, and `enthusiasm`, suggesting contextual co-occurrences related to labor topics. Despite stopword removal and lemmatization, the PMI approach effectively captured localized semantic associations. Relying on co-occurrence statistics rather than dense embeddings allowed for coherent and interpretable results, demonstrating the potential of unsupervised methods in specialized corpora such as financial news.

Target Word	Top Associated Words (PMI Score)
safety	uniglass (12.292), tolerability (12.292), corroborate (12.292), excellently (12.292)
goldman	underperform (13.671), sachs (13.671), niam (12.671), arrange (11.671)
mynet	nanjing (15.993), make (9.027)
ceiling	nonvisible (15.993), wall (14.408)
worldleading	manufacturer (10.086), oyj (8.080)
telecominvest	argument (14.993), megafon (12.993), find (10.600), currently (9.823)
eronen	svp (13.993), sami (12.993), senior (11.292)
priceshare	99483 (15.993), gratuitous (15.993)
korteniemi	anneli (15.993), 12 (10.378)
employee	15500 (9.600), whitecollar (9.600), society (9.600), 612 (9.600), enthusiasm (9.600)

Table 7: Top PMI-based associated terms for 10 randomly selected target words. Values in parentheses are PMI scores.

BONUS: PRE-TRAINED EMBEDDING AND RNN MODEL

For the bonus task, we implemented a recurrent neural network (RNN) using a bidirectional GRU architecture for sentiment classification on a financial news dataset. Input texts were encoded using pre-trained contextual embeddings from xlm-roberta-base, producing token-level vectors padded to a fixed length. The model comprises two GRU layers with dropout (0.3) and a final linear layer for binary or multiclass output, depending on the setup. We compared four variants: binary and multiclass classification, each with and without oversampling using RandomOverSampler to address class imbalance. Training was conducted on CPU using a stratified 70/30 train-validation split, AdamW optimizer (learning rate 1e-3), batch size 16, and cross-entropy loss over 10 epochs. Results show binary classification performed best, with 93% accuracy and macro-F1 of 0.92 under oversampling. The multiclass model achieved solid scores around 0.79 macro-F1, but oversampling slightly reduced performance—likely due to synthetic data noise. Overall, the GRU model outperformed classical models from Task 3 (Naive Bayes, FFNN) across all metrics. This underlines the strength of contextual embeddings and sequential architectures for capturing subtle sentiment patterns in financial text. Despite running solely on CPU, training was stable and efficient. Due to ISIS upload limits (max. 50 MB), we are unable to submit the full fine-tuned RoBERTa-based model. However, we are happy to provide the model and all associated files upon request.

Setup	Accuracy	Macro F1	Weighted F1	Validation Loss
Multi-class (no oversampling)	0.82	0.79	0.82	0.4406
Multi-class (oversampling)	0.81	0.78	0.81	0.6002
Binary (no oversampling)	0.92	0.90	0.92	0.2093
Binary (oversampling)	0.93	0.92	0.93	0.2241

Table 8: Evaluation Summary of GRU Model on Binary and Multiclass Tasks

REFERENCES

[1] Qurat Tul Ain, Mubashir Ali, Amna Riaz, Amna Noureen, Muhammad Kamran, Babar Hayat, and A Rehman. 2017. Sentiment analysis using deep learning techniques: a review. *International Journal of Advanced Computer Science and Applications* 8, 6 (2017).

[2] Md Shah Alam, Md Sabbir Hossain Mrida, and Md Atikur Rahman. 2025. Sentiment analysis in social media: How data science impacts public opinion knowledge integrates natural language processing (NLP) with artificial intelligence (AI). *American Journal of Scholarly Research and Innovation* 4, 01 (2025), 63–100.

[3] Desmond Ampaw-Asiedu, Eli Kofi Avickson, and Nicholas Nyonyoh. [n.d.]. Financial Forecasting and Planning: Predictive Models that Support Agile Financial Planning by Adjusting Forecasts Based on Emerging Trends and Macroeconomic Factors. *Journal homepage: www. ijrpr. com ISSN* 2582 ([n. d.]), 7421.

[4] Colm Kearney and Sha Liu. 2014. Textual sentiment in finance: A survey of methods and models. *International Review of Financial Analysis* 33 (2014), 171–185.

[5] Huina Mao, Scott Counts, and Johan Bollen. 2011. Predicting financial markets: Comparing survey, news, twitter and search engine data. *arXiv preprint arXiv:1112.1051* (2011).

[6] David Valle-Cruz, Vanessa Fernandez-Cortez, Asdrúbal López-Chau, and Rodrigo Sandoval-Almazán. 2022. Does twitter affect stock market decisions? financial sentiment analysis during pandemics: A comparative study of the h1n1 and the covid-19 periods. *Cognitive computation* 14, 1 (2022), 372–387.