Decoding Financial Sentiments: A Comparative Analysis of Machine Learning Approaches

Joy Wang (zw673)

Dec 12 2023

Abstract

In financial markets, rapid and accurate interpretation of news sentiment is crucial for effective decision-making. However, due to the use of domain-specific terminology, jargon, and slang, sentiment analysis of financial texts is challenging. The field of sentiment analysis in finance seeks to automate the process of discerning the positive, negative, or neutral nature of financial news. The project aims to determine the most effective techniques for financial sentiment analysis on a dataset of expert-labeled financial news sentences by comparing various feature extraction methods (bag of words, N-gram, TF-IDF, word embedding), traditional machine learning models (softmax regression, multinomial naive Bayes, etc.), and deep learning methods (neural networks and transformers). The study highlights four key points: 1) Simpler bag-of-words representations effectively capture key sentiment information in financial texts, while N-Gram and TF-IDF may introduce redundancy; 2) Traditional machine learning models underutilize the potential of complex features, with deep learning models benefiting more from richer representations and pre-trained word embeddings; 3) Addressing class imbalance is crucial; 4) FinBERT, pre-trained on financial data, outperforms traditional machine learning and generic BERT models, effectively capturing financial sentiment nuances. This work contributes to the broader effort of integrating advanced computational methods into financial decision-making processes.

1 Introduction

People who work in the financial markets are always keeping an eye on economic and financial news. They need to quickly figure out whether the news is positive or negative to make sound financial decisions. Using sentiment analysis models is a quick and easy way to get helpful information from the news. Financial sentiment analysis is the task of identifying and extracting opinions, and it is a widely used tool in the financial industry to inform investment decisions, gauge market sentiment, and identify potential risks and opportunities. However, financial sentiment analysis is a challenging task due to the use of domain-specific terminology, jargon, and slang. For example, in everyday language, ”share” means something good. But in finance, ”share” means a financial object or a stock, a neutral word. Additionally, the lack of large training data sets in specific domains is also a factor that limits the performance of text sentiment analysis models in the financial field. In this project, as detailed benchmarking on existing dataset, we aim to evaluate the performance of different sentiment analysis methods on financial data. We will use a publicly available dataset of financial news articles labeled by financial experts.

2 Background

To understand the context of this project, first, we need to understand what financial news texts cover. Financial markets include the trading of various assets such as stocks, bonds, foreign exchange, etc., and the text of financial news covers relevant information on these markets, such as company financial reports, economic indicators, and market trends. Financial texts usually contain a large number of professional terms, abbreviations, and vocabulary unique to the financial industry, which may be unfamiliar to general sentiment analysis models.

Secondly, we believe that a certain understanding of the basic concepts and steps of sentiment analysis is required. Sentiment analysis is a method that uses computer technology to automatically identify the emotional polarity contained in texts, which is often divided into positive, negative, and neutral. In financial texts, this means being able to judge the market impact of attitudes expressed in news reports or commentary. Text sentiment analysis can usually be further divided into three key steps: feature extraction, learning classification, and evaluation optimization. In feature extraction, text data is converted into numerical features that can be processed by machine learning models, using methods such as bag-of-words models and word embeddings. In the learning classification stage, an appropriate machine learning algorithm or deep learning model is selected and trained with training data to learn the classification of text emotions. After model training, use evaluation indicators such as accuracy, F1 score, etc. to evaluate the performance of the model. Based on the evaluation results, the model's hyperparameters can be adjusted or different feature extraction methods can be selected to improve the model's performance on new data. This step is iterative and model parameters can be adjusted multiple times to find optimal performance.

3 Setup

3.1 Dataset Overview

We utilized a polar sentiment dataset[1] consisting of sentences extracted from financial news articles. The dataset comprises 5842 sentences in the English language, categorized by sentiment. The categorization was based on the agreement rate of 5-8 annotators, indicating different levels of consensus among the annotators. The dataset consisted of two columns, ”sentence” and ”sentiment.” The ”sentiment” data has three types: neutral, positive, and negative, and we encoded them with the following rules.

Table 1: Sentiment Encoding

Sentiment	Code
neutral	0
positive	1
negative	-1

To evaluate the model's performance, the original dataset was divided into a training set, a development dataset, and a testing set with the ”train_test_split” function available in the ”sklearn.model_selection” module. We allocate 60% of the data to the training set, and the development set and test set each account for 20%. In the training set, the proportion of each sentiment is shown in Table 2 below.

[1] https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis/data

Table 2: Sentiment Distribution in the Training Set

Sentiment	Count	Percentage
1 (Positive)	1102	31%
-1 (Negative)	520	15%
0 (Neutral)	1883	54%

3.2 Environment

The experiments are conducted on Google Colab using Python 3.10.12. The hardware environment includes a Tesla T4 GPU, and the CPU is an Intel Xeon CPU. The Scikit-learn 1.2.2, PyTorch 2.1.0+cu118, gensim 4.3.2, tensorflow, and Hugging Face Transformers framework 4.35.2 were used during the training process.

4 Method

4.1 Feature Extraction Methods

In this part, we use techniques to represent text data as feature vectors.

4.1.1 Bag of Words

The bag-of-words model is a representation of text that disregards word order and focuses on the frequency of words in a document, treating each document as an unordered set of words. The main idea is to build a vocabulary of words appearing in the dataset and then to represent each sentence by a feature vector x whose length is the same as the vocabulary size, where $x\_i = 1$ means that the ith vocabulary word appears in this sentence, while $x\_i = 0$ otherwise. We can extract vocabulary based on the word frequency threshold, which means only including words that appear in at least the threshold number of different news sentences; this is important to avoid run-time memory issues and noisy features.

4.1.2 N-Gram Model

Beyond threshold-based vocabulary extraction, we apply the N-Gram Model to construct the ”bag of words.” The N-gram model is similar to the bag of words model, but instead of using individual words, we use N-grams, which are contiguous sequences of words. This article will mainly consider the 2-gram model.

4.1.3 TF-IDF Term Weighting

The Bag-of-Words model faces the feature-vanishing problem when common words with high corpus frequency dominate the feature space, diminishing the performance by obscuring more meaningful vocabulary. To address this, we adopt the TF-IDF (Term Frequency-Inverse Document Frequency) model, which assesses term importance in a document relative to a document collection. TF measures term frequency in a document, while IDF measures term significance across the document set by assigning lower weights to common words and higher weights to rare words. The TF-IDF formula is given by

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Where $TF(t, d)$ is the frequency of term t in document d, and $IDF(t, D)$ is the frequency of term t in the document set D.

4.1.4 Word Embedding

Word embedding is a technique in natural language processing that represents words as continuous vectors in a high-dimensional space, where the spatial relationships between vectors capture semantic similarities between words. Unlike traditional methods such as one-hot encoding, which represent words as sparse binary vectors, word embeddings encode semantic relationships and contextual information by placing similar words closer together in the vector space. This allows the model to capture the meaning of words based on their contextual usage in a given corpus. Popular word embedding models, like Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014), leverage large text corpora to learn these vector representations, enabling more nuanced and context-aware language understanding for various NLP tasks, including sentiment analysis, machine translation, and document clustering.

4.2 Traditional Classification Models

Based on the feature vectors we obtained in the last step, we try several different machine learning methods to classify sentiment in this part.

4.2.1 Softmax Regression

Softmax Regression is a generalization of logistic regression to handle multiple classes. The softmax function is applied to the output of a linear combination of input features and weights, converting them into probabilities. Given a set of classes, the softmax function normalizes the scores for each class, assigning probabilities to each class. The formula for the softmax function is:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where $P(y_i)$ is the probability of class i, z_i is the score for class i, and the denominator is the sum of the exponential scores for all classes. Softmax regression aims to minimize the cross-entropy loss, which measures the difference between predicted probabilities and the actual class labels.

4.2.2 Multinomial Naive Bayes

Multinomial Naive Bayes is a probabilistic model based on Bayes' theorem. In the context of text analysis, each feature represents the frequency of a word in a document. The ”naive” assumption of independence between features given the class label simplifies the computation. The model calculates the probability of a document belonging to a particular class using the product of the probabilities of each feature given that class, and then the class with the highest probability is assigned as the predicted class. The mathematical expression for Multinomial Naive Bayes is given by:

$$P(\text{class} = c|\text{doc}) \propto P(\text{class} = c) \prod_{i=1}^{n} P(\text{term}_i|\text{class} = c)^{count(\text{term}_i)}$$

Where $P(\text{class} = c|\text{doc})$ is the probability of the document (sentence) belonging to class c, $P(\text{term}_i|\text{class} = c)$ is the probability of term term_i given class c, and $count(\text{term}_i)$ is the count of term term_i in the document.

4.3 Deep Learning Methods

4.3.1 Convolutional Neural Network (CNN)

Based on the bag-of-words model or word embedding model, CNN uses one-dimensional convolutional layers to capture local features in the text. The convolution kernel slides over the input text, producing a series of feature maps that capture local information at different locations in the input text. This helps the model learn important patterns and features in the text. It also uses a pooling layer to extract the most salient information to reduce the dimensionality of feature maps. The pooled features are input into a fully connected layer with a softmax activation function, the learned feature mapping is associated with the emotion of the text, and the probability distribution of each emotion category is output. The emotion category predicted by the model is the category with the highest probability.

4.3.2 NLP Transformers

The pre-trained word embeddings often show good performance for NLP tasks because they retain the semantics and the syntax of the words in the sentence. However, they usually lack context-based mutability due to the one-to-one mapping. Therefore, we also try to improve the performance in sentiment analysis using advanced models - transformers, leveraging their ability to capture complex language patterns effectively.

In "Attention is All You Need", Vaswani et al. (2017) demonstrated a novel transformer architecture that transforms one sequence into another by using encoder and decoder, based on multi-head self-attention mechanisms.

BERT is a groundbreaking model introduced by Devlin et al., using a transformer architecture to understand the context of a word in relation to all the other words in a sentence, rather than in one direction. This bidirectional approach allows for a more nuanced understanding of language context and syntax.

DistilBERT is a smaller, faster, and more efficient version of BERT. It retains most of the performance of BERT while being more resource-efficient.

Created by Araci, FinBERT is a variant of BERT, pre-trained on a financial text corpus consisting of 1.8M news articles from the Reuters TRC2 dataset. It demonstrates the adaptability of transformer models to specialized domains, providing enhanced performance in NLP tasks in the financial field.

4.4 Evaluation Metrics

4.4.1 Accuracy

Accuracy is a crucial metric for assessing overall model correctness, particularly in multi-class classification tasks. On a dataset

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$$

accuracy is

$$acc(f) = \frac{1}{n} \sum_{i=1}^{n} I\{f(x^{(i)}) = y^{(i)}\}$$

where $I\{\cdot\}$ is an indicator function (equals 1 if its input is true and zero otherwise). However, accuracy can be misleading in imbalanced data sets, especially when the classes are unevenly distributed. So, we also need other metrics as supplements.

4.4.2 Precision and Recall

Precision measures how many of the samples predicted as positive by the model are actually positive; Recall refers to how many of all actual positive class samples are correctly predicted as positive classes.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Table 3: Confusion Matrix

	Predicted positive $\hat{y} = 1$	Predicted negative $\hat{y} = 0$
Positive class $y = 1$	True positive (TP)	False negative (FN)
Negative class $y = 0$	False positive (FP)	True negative (TN)

4.4.3 Macro-Averaged F1-Score

F1-Score combines precision and recall. In multi-class classification, using Macro or Micro-averaged F1-Scores provides insights into overall classification performance. Macro-averaging calculates the average F1-Score for each class, while Micro-averaging considers true positives, false positives, and false negatives for all classes. In the judgment of sentiment tendency of financial texts, whether positive, negative, or neutral emotions will affect the investment strategy, that is, the importance of each category is similar, and we do not want any one category to have an excessive impact on the overall evaluation. In this case, using the macro F1-Score is appropriate.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Macro-F1} = \frac{1}{N} \sum_{i=1}^{N} \text{F1-Score}_i$$

5 Experimental Analysis

5.1 Data Preprocessing

First, we checked the missing value in the dataset and found that there is not any missing value. Second, we convert all the words to lowercase as case normalization helps treat the same word in different cases as one, ensuring consistency. Third, Lemmatization is implemented because it converts words to their base or root form, which can be beneficial in understanding the underlying meaning and reducing the feature space. Finally, we strip punctuation, stop words, special characters, and extra white spaces since they often don't add significant meaning to the text, especially for this classification task. Removing punctuation can help reduce the noise in the data. The data samples before and after preprocessing are shown in Table 3.

Table 4: Processed Text Samples

Sentence	Sentiment	Processed Text
Finnish elevators and escalators maker KONE Corporation said on Tuesday (18 March) that it has received a major order from Sir Robert McAlpine to supply all elevators and escalators for the Watermark Place project in the City of London.	1	finnish elevator escalator maker kone corporation said tuesday march received major order sir robert mcalpine supply elevator escalator watermark place project city london
At 10.33 am, Huhtamaki was the market's biggest faller, 8.69 pct lower at 11.35 eur, while the OMX Helsinki 25 was 0.32 pct higher at 3,332.41, and the OMX Helsinki was up 0.47 pct at 11,687.32.	-1	huhtamaki market biggest faller pct lower eur omx helsinki pct higher omx helsinki pct
Nevertheless, the development can not be allowed to ruin the print newspaper, which continues to be Sanoma News' main medium.	0	nevertheless development allowed ruin print newspaper continues sanoma news main medium

We tokenize the processed text, compute the frequency distribution, and sort words by frequency. The top 100 frequently used words in the training set are visualized in Figure 1, in which the most frequently used words are "eur" meaning "Euros", "company", and "mn" meaning "million". There's a steep drop in frequency after the first few words.

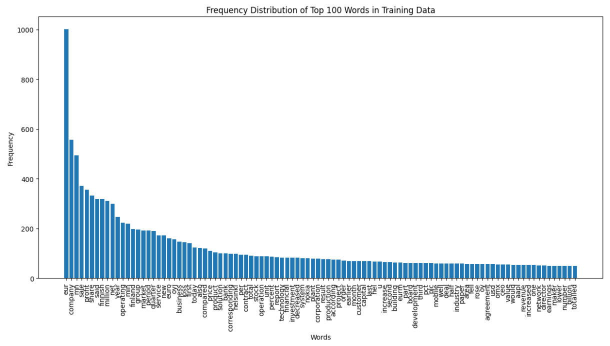


Figure 1: Frequency Distribution of Top 100 Words

5.2 Feature Extraction Experiment

5.2.1 Bag of Words

The threshold of the "bag of words" model is determined by the distribution of words based on the number of sentences they appear in. From Figure 2, we consider 2 as the threshold because it significantly reduces the number of words in the bag of words to reduce noise while it does not result in too few features.

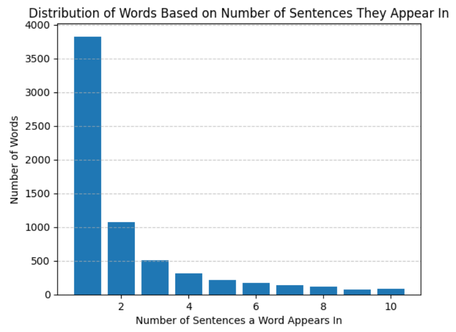


Figure 2: Distribution of Words on Different Thresholds

After deciding the threshold, we use the CountVectorizer function in "sklearn.feature_extraction.text" to vectorize the training set. The resulting vocabulary contains 1071 elements.

5.2.2 N-Gram

Using CountVectorizer, we set ngram_range=(1, 2), which means that the extracted features are single words (unigrams) and combinations of two words (bigrams). Similar to the threshold determination idea mentioned above, according to the chart, we think it is reasonable to ignore words with a document frequency lower than 2 (that is, min_df=2). The resulting n-gram vocabulary contains 7029 elements.

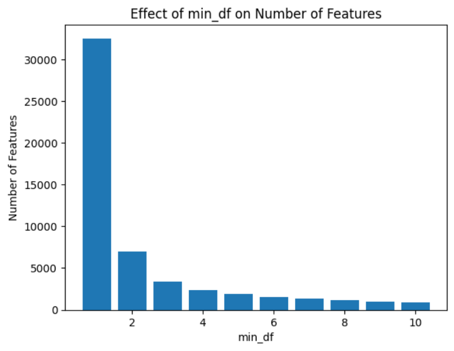


Figure 3: Distribution of Words on Different min_df

5.2.3 TF-IDF Term Weighting

We created a TF-IDF vectorizer with TfidfVectorizer in sklearn and used it to fit and transform the text data of the training and development sets. The number of features we got is 7151.

5.2.4 Pre-trained Word Embedding

We used the Word2Vec pre-trained model and converted text data into integer sequences for further use in a deep learning model. We downloaded the Word2Vec model (word2vec-google-news-300) using Gensim's API. Then we tokenize Text Data into lists of words. The Keras Tokenizer was employed to convert the raw text data into integer sequences. Then, the integer sequences were padded to ensure that all sequences had the same length (50 in this case). The resulting train_padded and dev_padded are ready to be used as input to a deep learning model, such as an embedding layer in a neural network.

Due to limitations in computing power, we only load the word vectors of the first 300,000 words in the model (about 10% of the complete model), covering about 64% of the words in the training text. And we only selected the first 50 dimensions of the word vector, that is, embedding_dim=50.

5.3 Training, Tuning and Results

5.3.1 Softmax Regression

In softmax regression classification, the libraries we use are sklearn and LogisticsRegression. We set the hyperparameter C to different values as in Fig 4. The smaller the value, the stronger the regularization strength. We chose the L-BFGS quasi-Newton method as the solver with L2 and no regularization and "liblinear" as the solver with L1 regularization. When there is no regularization, the accuracy on the training set is 0.936 while 0.415 on the development set, showing an obvious overfitting. According to Fig 4, the model with L1 regularization performs best with C=0.01, resulting accuracy of 0.537 on the training set and 0.527 on the development set. Accordingly, macro F1 Scores are 0.233 and 0.230. Moreover, we found that using features extracted by N-gram and TF-IDF did not significantly improve the prediction effect of the model.

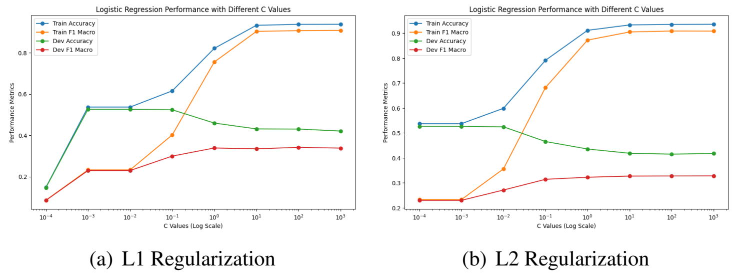
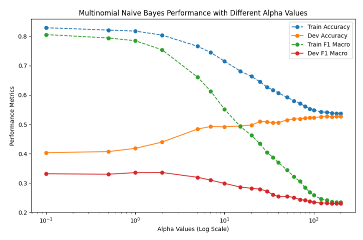

(a) L1 Regularization (b) L2 Regularization

Figure 4: Model Performance with Different C

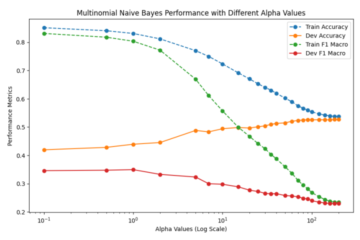
Finally, we united training and validation sets as one training set, used bag of words and a softmax regression model with L1 regularization and C of 0.01, and obtained the final results on the test set: accuracy on the train set is 0.535 and on test set is 0.541; macro F1 score on the train set is 0.232 and on test set is 0.234.

5.3.2 Multinomial Naive Bayes

We implement a text classification workflow using a Multinomial Naive Bayes model for training and evaluating both training and development datasets. First, import necessary modules from scikit-learn (MultinomialNB for Naive Bayes classifier and metrics for evaluation). Besides, we tune hyperparameter alpha for Laplace Smoothing. The larger the alpha, the stronger the smoothing effect, because it is equivalent to adding more hypothetical data points to the estimate, reducing the influence of the original data. In experiments, we found that when alpha is 100 or above, the model has strong generalization ability, and the accuracy on the training set and development set reach 0.548 and 0.522 while macro F1 Scores are 0.258 and 0.234. We also found that using features extracted by N-gram and TF-IDF got almost the same performance as using bag of words.



(a) Bag of Words



(b) N-Gram

Figure 5: Model Performance with Different Alpha

Finally, we united training and validation sets as one training set, used bag of words and a Multinomial Naive Bayes model with an alpha of 100, and obtained the final results on the test set: accuracy on the train set is 0.540 and on test set is 0.544; macro F1 score on the train set is 0.245 and on test set is 0.242.

5.3.3 Word Embedding and CNN

During model training, we found that because some categories have more data, the model tends to classify most of the data into the same category. To solve this problem, we used RandomOverSampler in the imblearn library to perform random over-sampling to address class imbalance.

We established a neural network for text classification in TensorFlow. The model is configured with specific parameters: num_classes is set to 3 for a three-class classification task, and max_length is defined as 50, indicating the maximum sequence length. The neural network architecture comprises an embedding layer initialized with pre-trained Word2Vec weights, a 1D convolutional layer with 64 filters, kernel size 3, ReLU activation, and L2 regularization, followed by global max-pooling, a dense layer with 64 units and ReLU activation, and a dropout layer with a rate of 0.5. The output layer employs softmax activation for multi-class classification. The model is compiled using a custom Adam optimizer with a learning rate of 0.001, sparse categorical crossentropy loss, and accuracy as a metric. To address class imbalance, class weights are computed using compute_class_weight, and training employs an early stopping callback with a patience of 20 epochs. Then, conducted the computation of F1 scores on the training and validation sets using scikit-learn's f1_score function.

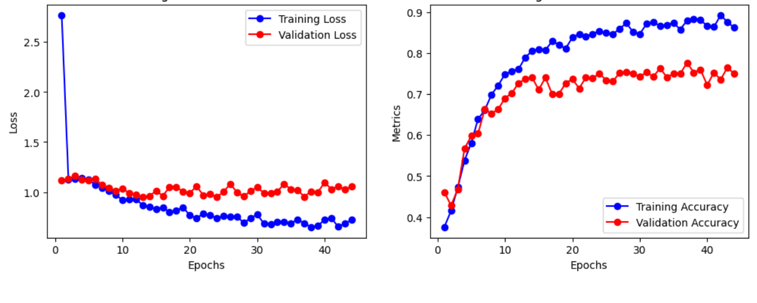


Figure 6: Learning Curves

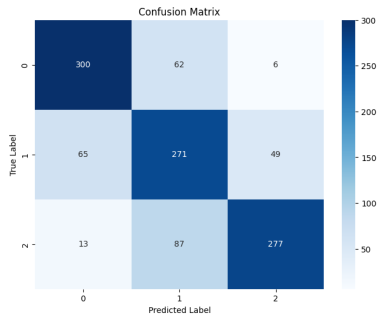


Figure 7: Confusion Matrix

We believe that the model effect in the 12th epoch is relatively better because the subsequent models have a certain degree of overfitting. In the 12th epoch, the prediction accuracy of the model on the training set was 0.761, and the loss was 0.932. The prediction accuracy on the validation set was 0.727 and the loss was 0.972. Finally, we got an accuracy of 0.595 and a macro F1 score of 0.503 on the test set.

The result is significantly better than using Softmax Regression and Multinomial Naive Bayes, as accuracy increased to an extent, and the macro F1 score improved noticeably. We noticed that accuracy might not be a reliable metric when dealing with imbalanced datasets, as a high accuracy could be achieved by simply predicting the majority class. The macro F1 score considers precision and recall across all classes, making it more informative in scenarios with imbalanced class distributions. An increase in macro F1 score suggests that the model is now better at correctly classifying instances from the minority classes. However, this model still has room for improvement, because according to the results, it may be overfitting on the validation set.

5.3.4 FinBERT

We tried BERT, distilBERT, and FinBERT, among which FinBERT got the best results on the test set, so this section will mainly discuss how to use FinBERT for sentiment classification of financial texts. First, we installed the necessary libraries, including FinBERT[2] and its dependencies, using the pip install command. After importing relevant libraries and loading the pre-trained FinBERT model and tokenizer, a text classification pipeline is established. Subsequently, the model is applied to a test dataset (test_data) for sentiment analysis. The script then maps the model's output labels ('Negative', 'Neutral', 'Positive') to numerical values (-1, 0, 1) and calculates the accuracy of the predictions on the test dataset by comparing them to the true labels. The final output is the test accuracy of approximately 0.654 and the macro F1 score of 0.559. We found that fine-tuning FinBERT using our dataset did not improve predictions. Possibly because our fine-tuning dataset is relatively small, this may lead to overfitting, especially if the pre-trained model already contains a large amount of information.

5.3.5 General Observation

First, the lack of significant performance differences between bag-of-words, N-Gram, and TF-IDF representations on traditional machine learning models (e.g., logistic regression, Naive Bayes) may be due to the nature of financial text that makes certain information content of language features reduced. Key information for sentiment analysis in financial texts may be well captured by simpler bag-of-words representations, while features extracted from N-Gram and TF-IDF may contain redundant or highly relevant information.

Second, traditional machine learning models may not fully exploit the potential of more complex feature representations. Deep learning models

[2]https://github.com/yya518/FinBERT.git

(such as the CNN and FinBERT in the analysis) may benefit more from richer representations. Word embeddings contribute significantly to the performance of CNN models, possibly because Pre-trained word embeddings provide a form of transfer learning. The embeddings have been learned on large and diverse datasets, and the knowledge gained from this training can be leveraged by the CNN model for the specific task of sentiment analysis on financial texts. This is particularly useful when the labeled dataset for the target task is limited. Also, word embeddings represent words as continuous vectors in a high-dimensional space. This continuous representation allows the model to capture more nuanced relationships between words and their contextual meanings compared to discrete representations like one-hot encoding or Bag of Words.

Third, Handling class imbalance is crucial, as demonstrated by the CNN model's improvement after using RandomOverSampler.

Finally, FinBERT, being pre-trained on financial data, demonstrates superior performance compared to traditional machine learning models and generic BERT models. It appears to capture financial sentiment nuances effectively.

6 Discussion and Prior Work

In our exploration of sentiment analysis within the broader context of natural language processing, we draw inspiration from foundational works that have paved the way for understanding and categorizing sentiment in text. Pang and Lee's seminal work, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales" (2005), introduced a novel approach by leveraging class relationships inherent in human-annotated rating scales. Extending beyond this, our work takes a leap into the domain-specific realm of financial sentiment analysis. Navigating the challenges posed by the intricacies of financial jargon and terminology, our research showcases the adaptability of sentiment analysis models to specialized domains.

Another influential reference is Turney and Littman's "Measuring Praise and Criticism: Inference of Semantic Orientation from Association" (2003), which introduced the concept of pointwise mutual information for determining semantic orientation. Building upon this foundation, our work adapts and enhances sentiment analysis methodologies, specifically tailoring them for the unique linguistic nuances of the financial domain.

In the realm of financial sentiment analysis, traditional machine learning models have been widely employed. A study by Ding et al. (2014) titled "Using Financial News to Predict Stock Price Movements" can be referenced. This paper explores the use of bag-of-words models and sentiment analysis to predict stock price movements. Our work goes beyond traditional methods by incorporating advanced feature extraction techniques like word embeddings and leveraging deep learning models for enhanced performance.

Our work introduces advanced deep learning models, such as Convolutional Neural Networks (CNNs) and transformers, into financial sentiment analysis. Prior research in sentiment analysis often focuses on simpler models, and our approach aligns with the growing trend of using more advanced models for text analysis tasks. For instance, we draw inspiration from Vaswani et al.'s "Attention is All You Need" (2017) and Devlin et al.'s "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (2019). The incorporation of transformer models, including FinBERT tailored for financial texts, demonstrates a paradigm shift towards more sophisticated language understanding in the financial domain.

Our comparative analysis showcases the superiority of deep learning models over traditional machine learning methods in financial sentiment analysis. The iterative tuning of hyperparameters and feature extraction methods demonstrates a commitment to finding optimal performance. The transition from Softmax Regression and Multinomial Naive Bayes to CNNs and transformers illustrates the continuous pursuit of improved accuracy and robustness. The use of FinBERT further solidifies our commitment to leveraging state-of-the-art models for domain-specific tasks.

Looking ahead, our work opens avenues for future research in several directions. Firstly, exploring ensembles of models, combining the strengths of different architectures, could be considered for even more robust sentiment analysis. Additionally, real-time sentiment tracking and adaptation of models to dynamic financial landscapes present intriguing challenges for future investigations. Incorporating external knowledge sources, such as market data or global events, could further enhance the contextual understanding of financial sentiment. Finally, there's room for exploring interpretability in deep learning models, making the decision-making process more transparent and understandable for stakeholders.

7 Conclusion

In this research, we have successfully showcased the power of machine learning in financial sentiment analysis. We have been able to extract nuanced sentiments from complex financial texts, providing deep insights into market trends and investor behavior. This project not only highlights the potential of machine learning in transforming financial analysis but also sets the stage for future research in real-time sentiment tracking and the incorporation of diverse data sources. The broader implications of this work are substantial, offering financial analysts, investors, and policymakers new tools for data-driven decision-making in an increasingly dynamic economic landscape. The implications of this work extend to investors, analysts, and financial institutions seeking data-driven insights for informed decision-making.

Reference:

Araci, D. (2019). FinBERT: Financial Sentiment Analysis with Pre-trained Language Models. *ArXiv, abs/1908.10063.*

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *North American Chapter of the Association for Computational Linguistics.*

Ding, X., Liu, B., & Zhang, L. (2014). Using financial news to predict stock price movements. *Journal Name, Volume(Issue).*

Huang, Allen H., Hui Wang, and Yi Yang. (2022). FinBERT: A Large Language Model for Extracting Information from Financial Text. *Contemporary Accounting Research.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (pp. 3111–3119). Curran Associates Inc.

Pang, A., & Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *Journal Name, Volume(Issue).*

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics.

Turney, P. D., & Littman, M. L. (2003). Measuring praise and criticism: Inference of semantic orientation from association. *Journal Name, Volume(Issue)*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).

Yang, Yi, Mark Christopher Siy Uy, and Allen Huang. (2020). Finbert: A pretrained language model for financial communications. *arXiv preprint arXiv:2006.08097*.