INF8225 - Financial sentiment analysis on news and tweets

Mathéo Benoit-Paraschivoiu , **Jean Siffert** and **Justin St-Arnaud**
Polytechnique Montréal
{matheo.benoit-paraschivoiu, jean.siffert, justin.st-arnaud}@polymtl.ca

Abstract

This paper introduces various models and evaluates their efficacy in a financial sentence classification task, which involves categorizing sentences and tweets into negative, neutral, or positive sentiments. The implemented models are subsequently compared with pre-trained ones. While the optimal performance is attained with pre-trained classification models, our in-house classifiers, developed and trained within the constraints of limited resources over the course of a month-long project, demonstrate good performance and yield compelling results, only slightly worse than completely pre-trained classification models (-3.5%).

1 Introduction

1.1 Problem

Financial text data, including news articles, analyst reports, company public announcements, and social media posts, frequently employ specialized terminology and ambiguous language. Not only these text data can be challenging to interpret, they come in large volumes because of the dynamic nature of the financial industry.

Financial sentiment analysis emerges as an important tool in this context, capable of extracting valuable insights. Indeed, one could ask himself if a certain financial news or sentence is negative, neutral or positive. In this report, we explore the possibility of training a model capable of identifying the sentiment of a financial sentence.

1.2 Literature review

We examined various studies and approaches that have been proposed to tackle financial sentiment analysis tasks. One notable paper that stood out in our investigation is "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models" [Araci, 2019]. This study proposes a model called FinBERT capable of tackling this classification task.

FinBERT is a language model based on BERT (Bidirectional Encoder Representations from Transformers), which has been further pre-trained on a large financial corpus. The author demonstrate that FinBERT significantly improves the performance on two financial sentiment analysis datasets, FiQA sentiment scoring and Financial PhraseBank, achieving state-of-the-art results at the time it was written.

The study also explores different aspects of the model, including the effects of further pre-training on a financial corpus and fine-tuning only a small subset of model layers to decrease training time without a significant drop in performance.

This paper highlights the potential of pre-trained language models in addressing the unique challenges of the financial domain, but we observed that the study focused solely on two specific datasets. Consequently, we thought interesting of developing models capable of further generalizing across various datasets. We also expanded our investigation to incorporate a diverse range of techniques, going from more classic machine learning models such as Bidirectional LSTMs and Encoder-Only Transformers to pre-trained models.

2 Methods

We have implemented 5 types of models to classify the sentiment of finance-related sentences. This section describes the pre-processing operations and the theoretical foundations of the 5 model types.

2.1 Pre-processing

Our dataset is a combination of 4 datasets found on HuggingFace [Malo *et al.*, 2014], [nickmuchi, 2022], [TimKoornstra, 2024], [chiapudding, 2023]. The datasets used are shown in the table 1 below, alongside with the number of samples in each one.

Table 1: Datasets used

Dataset	Samples
`financial_phrasebank/sentences_allagree`	2260
`nickmuchi/financial-classification`	5057
`TimKoornstra/financial-tweets-sentiment`	38091
`chiapudding/kaggle-financial-sentiment`	5842

We carried out several pre-processing steps to standardize the datasets and remove duplicates and non-compliant samples. We obtained a total of **43833** samples in our dataset. Subsequently, we replaced highly variable symbols such as URL links, profile indicators (e.g. @ on Twitter) and symbols representing quoted stocks (tickers) with predefined to-

kens; <URL>, <PROFILE> and <TICKER> respectively. This allowed us to reduce the size of the vocabulary, despite some loss of semantic information. The dataset was then split in *train*, *validation* and *test* sets containing respectively 35066, 4383 and 4384 samples following a standard 80-10-10% split. The sentences are then passed through a tokenizer that varies according to the model. Apart for the pre-trained models, all models end with a classification head to obtain a 3-element vector for the `Negative`, `Neutral` and `Positive` classes.

2.2 Bidirectional LSTM

Long Short-Term Memory networks (LSTMs) [Hochreiter and Schmidhuber, 1997] are a type of Recurrent Neural Network (RNN) specifically designed to overcome the problem of learning long-term dependencies and mitigate the issue of vanishing gradient. Recurrent neural networks are well-suited to operate on sequences of data, and are therefore very appropriate for tackling sentiment classification in sentences. To better capture the relationship between each word in the sentence, we implemented bidirectional LSTM, meaning that the input is processed in both forward and backward directions, and the results are combined to obtain enriched information [Dive into Deep Learning, 2023]. The figure 1 below depicts the general architecture of a bidirectional RNN.

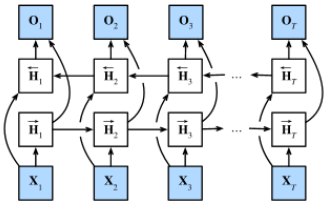


Figure 1: Bidirectional RNN [Dive into Deep Learning, 2023]

Our network implements a bidirectional LSTM that receives tokenized sentences as input, transforms them to embeddings and computes the hidden states sequentially. Afterwards, we build a feature vector that will be used to classify the input by combining the last hidden states of the forward and backward pass, thus obtaining a vector of size $2 \times \text{hidden\_size}$, as suggested in [Analytics Vidhya, 2020]. This vector is then passed through a fully connected network to classify inputs among the 3 labels (negative, neutral, positive).

2.3 Encoder-Only Transformer

The architecture of transformers was introduced in the 2017 article "Attention Is All You Need" [Vaswani *et al.*, 2023]. Transformers leverage the attention mechanism and consist of 2 main parts: an encoder and a decoder. For this classification task, we will rely solely on an encoder. The model takes tokenized sentences as input, transforms them into embeddings and adds positional encoding to take into account the relative position of tokens within the sentence. The data is then passed to the encoder, which leverage the attention mechanism to add information relative to the importance of each word with respect to the others. Afterwards, we produce a feature vector for the classification head, which is an average of the encoder outputs weighted by their distribution (using a `softmax` function). This vector is then passed through a fully connected network to perform classification.

2.4 Fine-tuned BERT Encoder

BERT, short for Bidirectional Encoder Representations from Transformers, is a state of the art pre-trained language model developed by Google introduced in 2018 [Devlin *et al.*, 2019]. It has been pre-trained on the Toronto BookCorpus (800M words) and English Wikipedia (2,500M words) corpus. It's pre-training consists of two tasks: Firstly, 15% of tokens are randomly masked, and the model is trained to predict these masked tokens based on the surrounding context. Secondly, it learns to determine whether two sentences follow each other in a text or not. BERT comes in two sizes: `BERTBASE`, consisting of 12 encoders with 12 bidirectional self-attention heads, totalling 110 million parameters, and `BERTLARGE`, with 24 encoders and 16 bidirectional self-attention heads, totalling 340 million parameters.

BERT's architecture can be broken down into three main steps. First, sentences are tokenized using the WordPiece tokenizer [Wu *et al.*, 2016], resulting in a vocabulary of 30,000 elements. Second, these tokens are passed through an embedding layer, converting them from one-hot encoded tokens into numerical representations. Finally, the embeddings undergo processing by a stack of encoders based on the transformer architecture explained above.

In our project, we extract embeddings from the final layer and employ pooling, focusing on the `[CLS]` token. These pooled embeddings are then fed into a two-layer neural network acting as a classification head, which maps the embeddings to three nodes for classification purposes.

2.5 Stacked model

During the project, we had the intuition to leverage our knowledge of emojis used in tweets. Indeed, certain emojis such as rockets, upward-trending stocks, moons, and others carry strong bullish connotations. Therefore, we opted to develop an additional model, a type of stacked model. This model would utilize the predictions from the previously fine-tuned BERT model and would simply add an emoji layer to the predictions, automatically classifying a sentence into the positive class if it encountered any bullish emoji.

2.6 Pre-trained models

In recent years, there has been a remarkable democratization of powerful large language models, largely thanks to platforms like Hugging Face. This platform enables users to access pre-trained models and fine-tune them for specific tasks to achieve impressive results. This accessibility empowers regular users who may not have the resources or expertise to train such models themselves.

In our project, we opted to explore some of these models available on the Hugging Face Model Hub, specifically focusing on models designed for sentence classification. We selected four models:

- `distilbert-base-cased`

- `bert-base-cased`
- `bertweet-base-sentiment-analysis`
- `distilroberta-finetuned-financial-news-sentiment-analysis`

The first two models are similar to our previous model. They utilize BERT to generate embeddings and then incorporate a classification head. DistilBERT [Sanh *et al.*, 2020] is a lighter version of the BERT model, aiming to maintain BERT's performance while reducing its size.

The third model has been pre-trained on tweets [Pérez *et al.*, 2021]. Given the abundance of tweets in our dataset, we decided to leverage this model for its specialized training on tweet data.

Lastly, the fourth model is a financial sentence classifier [Romero, 2024]. We included this model to compare against our own models, as it is pre-trained on a similar task, enabling us to assess our performance relative to an established benchmark in the financial domain.

3 Experiences

3.1 Bidirectional LSTM

In terms of pre-processing, we decided to use the `basic_english` tokenizer from the `torchtext` module, as this is PyTorch's default tokenizer and works only in English, which is our case [PyTorch, 2024b]. We then performed a grid search on several hyperparameters, analyzing performance on the validation set to find the optimal combination. The table 2 below shows the final hyperparameters values.

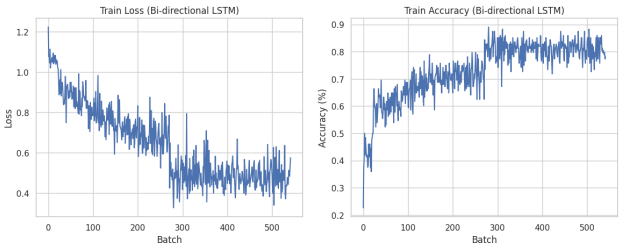
Table 2: Hyperparameters for Bidirectional LSTM

Hyperparameter	Value
`max_sequence_length`	64
`min_frequency`	2
`batch_size`	128
`dim_embeddings`	16
`dim_hidden`	32
`n_layers`	2
`dropout`	0.1
`learning_rate`	0.01
`epochs`	2
`clip_grad_norm`	1.0

We achieved satisfactory performance with an accuracy of 73.74% on the test set. Figure 2 illustrates the graphs of loss and accuracy during training. Both metrics have a fairly high variance, and the 2 epochs are clearly distinguishable.

3.2 Encoder-Only Transformer

We have also used the `basic_english` tokenizer from the `torchtext` module. We followed the same protocol as for LSTM, performing a grid search to find the optimal combination of hyperparameters. Interestingly, the resulting network is very small, with only one encoder layer, embeddings of dimension 16 and hidden layers of dimension 32. Indeed, the model rapidly overfitted when increasing its size, leading to poor results. The table 3 below shows the final hyperparameters values.



(a) Loss (b) Accuracy (%)

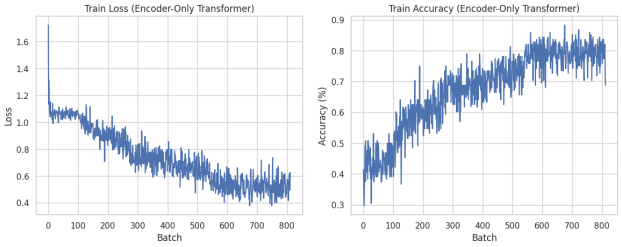
Figure 2: Bidirectional LSTM - Training loss and accuracy

In addition, several approaches were tried to construct the feature vector passed to the classification head, such as taking only the information from the first token or averaging the outputs. We also tried to average the weighted outputs. We computed the output distribution with a `softmax` function, then summed the outputs multiplied by the distribution. In the end, this strategy delivered the best performance.

Table 3: Hyperparameters for Encoder-Only Transformer

Hyperparameter	Value
`max_sequence_length`	64
`min_frequency`	2
`batch_size`	128
`dim_embeddings`	16
`dim_hidden`	32
`n_layers`	2
`n_heads`	4
`dropout`	0.15
`learning_rate`	0.01
`epochs`	3
`clip_grad_norm`	1.0

We achieved satisfactory performance with an accuracy of 73.71% on the test set, slightly lower than the previous 73.74% attained by LSTM. This result is unexpected, as we thought that the attention mechanism would produce better results. Figure 3 illustrates the graphs of loss and accuracy during training.



(a) Loss (b) Accuracy (%)

Figure 3: Encoder-Only Transformer - Training loss and accuracy

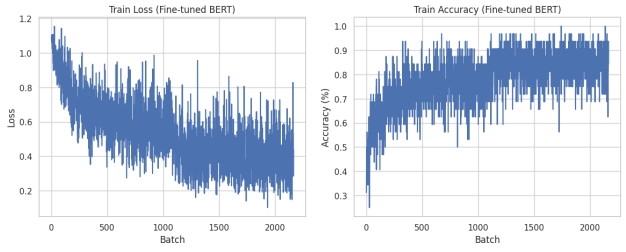
3.3 Fine-tuned BERT Encoder

To implement our classifier utilizing BERT embeddings, we fine-tuned `BERTBASE`. Fine-tuning `BERTLARGE`, even for a subset of layers, was too resource-intensive and did not significantly improve performance considering the added computation time. We fine-tuned BERT on our training dataset across all of its layers. The table 4 below shows the final hyperparameters values. We used only 2 epochs to prevent overfitting given the size of the model. Other hyperparameters, such as the dimensions of the embeddings and the hidden size, were set to the default values for BERT.

Figure 4 illustrates the graphs of loss and accuracy during training.

Table 4: Hyperparameters for fine-tuned BERT classifier

Hyperparameter	Value
`max_sequence_length`	64
`batch_size`	32
`learning_rate`	2e-5
`epochs`	2
`clip_grad_norm`	1.0



(a) Loss (b) Accuracy (%)

Figure 4: Fine-tuned BERT encoder - Training loss and accuracy

The BERT model demonstrates significant improvement, achieving an accuracy of 79.55%, which is approximately a 6% enhancement over our previous results. This good performance highlights the effectiveness of pre-training, resulting in clearly better embeddings compared to our previous encoder, which was a similar model but without pre-training.

3.4 Stacked model

Data exploration led us to designate the following emojis as bullish: Rocket, Chart Increasing, Chart Increasing with Yen, Fire, Money Bag, Gem Stone, Dollar Banknote, Money With Wings, Full Moon, Crescent Moon, First, Quarter Moon, Waxing Gibbous Moon, Full Moon with Face, Sun with Face and Full Moon.

Subsequently, by classifying any sentence containing one of these emojis as positive in the training set, we attained an accuracy of 94.6%, on the subset of sentences containing these emojis. Encouraged by this promising result and this high confidence level, we proceeded to evaluate the true stacked model on the test set, following the theory previously explained. When doing this, the stacked model achieved an accuracy of 79.41%, below the fine-tuned BERT model without the stacking.

3.5 Pre-trained models

Using Hugging Face pretrained models was fairly simple. We created a generic class that served as a trainer for all models. It charged the model's tokenizer from the hub, then the model itself, and simply used Hugging Face's Trainer class to train the model. We fine-tuned these models on our training set. Most of the hyperparameters are set by the models themselves. For the remaining ones, below are the ones we used. We employed a very small batch size since our Colab environment's GPU could not handle larger batch sizes due to RAM limitations. The table 5 below shows the final hyperparameters values.

Table 5: Hyperparameters for pre-trained models

Hyperparameter	Value
`batch_size`	16
`learning_rate`	2e-5
`epochs`	2
`weight_decay`	0.01

3.6 Code

The code containing the implementations of the various models as well as the realizations of the experiments mentioned is accessible on GitHub at the following address : GitHub repository.

4 Results

Overall, we trained a total of 8 models, including 4 implemented by ourselves and 4 pre-trained models. The latter performed best, and it seems that their deep understanding of the language is a real strength for semantic representation and classification. Fine-tuned BERT encoder performed similarly, and also benefited from pre-training. Fully self-trained models offer lower performance, at around 73.7%. Our attempt to use our knowledge of emojis proved unsuccessful, not helping our model's performance. The reason for that is that our fine-tuned BERT model already performed very well on the subset of sentences containing emojis, achieving an impressive accuracy of 94.68%. Overall, the best model is therefore `bertweet-base-sentiment-analysis` with an accuracy of **83.03%** on the test set. Table 6 summarizes the accuracy of each model on the test set.

5 Critical analysis

In our project, we used a variety of resources to understand the subjects and technologies involved. Firstly, we applied theoretical knowledge gained from course lectures [Pal, 2024], particularly on concepts like LSTMs and transformers. We also leveraged practical experience, including the code and experiences applied with transformers in lab 3, to help us with this part of the project. Additionally, our past courses

Table 6: Performance of models on the test set

Model	Accuracy (%)
Bi-LSTM	73.74
Encoder-Only Transformer	73.71
Fine-tuned BERT Encoder	79.55
`distilbert-base-cased`	79.26
`bert-base-cased`	80.43
`finiteautomata/bertweet-base-sentiment-analysis`	**83.03**
`mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis`	80.57

at Polytechnique Montréal, such as *INF8460-Traitement automatique de la langue naturelle* [Zouaq, 2024], provided valuable insights into BERT and pre-trained models.

Scientific papers detailing the models used in our project served as crucial references and helped us guide some of our experimentation [Vaswani *et al.*, 2023][Hochreiter and Schmidhuber, 1997][Devlin *et al.*, 2019][Araci, 2019]. Furthermore, we heavily relied on online blogs and tutorials for sentence classification, leveraging their guidance to implement models in code effectively [Singh, 2021][Analytics Vidhya, 2020]. Finally, the PyTorch documentation proved to be a very useful resource, offering detailed explanations and examples to navigate through the implementation process [PyTorch, 2024a].

These resources collectively enabled us to implement, train, and evaluate our models effectively. Overall, we are satisfied with the performance of our models. Despite facing stiff competition and working within constraints of limited resources and time, our best-performing classification model achieved results only slightly worse than completely pre-trained classification models (-3.5%), although using BERT to generate embeddings. Also, the financial classification pre-trained model only performed 1% better than our own models, which is encouraging.

Moving forward, we acknowledge potential avenues for improvement. Further hyperparameters tuning and better data exploration could have potentially improved our results. Taking a look at our models mistakes, some of which proved challenging even for human detection, we recognize the potential benefits of refining dataset selection and better leveraging the nuances in tweet data. Despite these potential improvements, we take pride in our achievements, demonstrating relatively comparable results with minimal time and resources at our disposal.

6 Conclusion

In conclusion, we implemented 4 classifiers in this project: a LSTM, a transformer (encoder only), a BERT-based classifier, and a stacked model based on emoji intuition. Additionally, we fine-tuned some pre-trained models and assessed their performance on our test set. We achieved an accuracy of 79.55% with our fine-tuned BERT model, while the best overall model was `bertweet-base-sentiment-analysis`, scoring 83.03%. We are impressed with the performance of the pre-trained models and are pleased with our own, which only differ by a few percentage points.

References

[Analytics Vidhya, 2020] Analytics Vidhya. First Text Classification in PyTorch. https://www.analyticsvidhya.com/blog/2020/01/first-text-classification-in-pytorch/, 2020. [Online; accessed April 24, 2024].

[Araci, 2019] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models, 2019.

[chiapudding, 2023] chiapudding. Kaggle financial sentiment dataset. https://huggingface.co/datasets/chiapudding/kaggle-financial-sentiment, 2023. Accessed: April 24, 2024.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[Dive into Deep Learning, 2023] Dive into Deep Learning. Bidirectional Recurrent Neural Networks. https://d2l.ai/chapter_recurrent-modern/bi-rnn.html, 2023. [Online; accessed April 24, 2024].

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[Malo *et al.*, 2014] P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala. Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65, 2014.

[nickmuchi, 2022] nickmuchi. Financial classification dataset. https://huggingface.co/datasets/nickmuchi/financial-classification, 2022. Accessed: April 24, 2024.

[Pal, 2024] Christopher J. Pal. I.a.:tech. probabilistes et d'apprentissage. Course, I.A.:tech. probabilistes et d'apprentissage, Université Polytechnique Montréal, 2024.

[PyTorch, 2024a] PyTorch. Pytorch documentation. https://pytorch.org/docs/stable/index.html, 2024.

[PyTorch, 2024b] PyTorch. Pytorch text data utilities. https://pytorch.org/text/stable/data_utils.html, 2024. Accessed: April 24, 2024.

[Pérez *et al.*, 2021] Juan Manuel Pérez, Juan Carlos Giudici, and Franco Luque. pysentimiento: A python toolkit for sentiment analysis and socialnlp tasks, 2021.

[Romero, 2024] Manuel Romero. Mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis · hugging face. https://huggingface.co/mrm8488/distilroberta-finetuned-financial-news-sentiment-analysis, 2024.

[Sanh *et al.*, 2020] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[Singh, 2021] Ankur Singh. Bert fine-tuning tutorial with pytorch. https://colab.research.google.com/github/Ankur3107/colab_notebooks/blob/master/classification/BERT_Fine_Tuning_Sentence_Classification_v2.ipynb#scrollTo=EKOTlwcmxmej, 2021.

[TimKoornstra, 2024] TimKoornstra. Financial tweets sentiment dataset. https://huggingface.co/datasets/TimKoornstra/financial-tweets-sentiment, 2024. Accessed: April 24, 2024.

[Vaswani *et al.*, 2023] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[Wu *et al.*, 2016] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[Zouaq, 2024] Amal Zouaq. Traitement automatique de la langue naturelle. Course, Traitement automatique de la langue naturelle, Université Polytechnique Montréal, 2024.