BOSTON UNIVERSITY

BU Questrom School of Business

Mutual Fund Style Classification

Team Members and Student ID： Jiayang Xu （U91377797）

Ziyuan Zhang（U24066039）

Contents

Executive Summary

Goal:

This study focuses on predicting the investment strategies of mutual funds based on their text summaries, with the primary objective of exploring the application of natural language processing (NLP) techniques in this context. By leveraging deep learning models, specifically Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), we aim to automate the analysis of mutual fund prospectuses and provide valuable insights for informed decision-making. Through this research, we delve into the potential of NLP in extracting meaningful information from financial text data and demonstrate its effectiveness in predicting investment strategies.

Methodology:

1. Data: Utilized two text corpora - MutualFundSummary (containing more than 500 mutual fund summaries) and MutualFundLabels.csv (containing information on 467 mutual funds). Processed and matched the data, creating training, validation, and test sets.

2. Text Vectorization: Employed two approaches for converting text summaries into vector representations: (a) self-trained Word2Vec models with feature selection using a knowledge base, and (b) pre-trained BERT embeddings.

3. Models: Trained and evaluated CNN and RNN models with varying architectures and hyperparameters.

Main Findings:

1. BERT embeddings outperformed Word2Vec models in terms of prediction accuracy.

2. Among feature-selected data, the RNN model exhibited the best performance. With BERT embeddings, the CNN_Large model achieved the highest metrics on the validation set (F1-score: 0.85, AUC: 0.8568, accuracy score: 0.8495).

3. Larger models demonstrated superior performance with BERT embeddings but did not significantly improve performance with feature-selected data.

Conclusion:

This study demonstrates the effectiveness of applying deep learning models for predicting mutual fund investment strategies from text summaries. Further research is needed to enhance model interpretability and validate performance on larger datasets.

Recommendations:

1. Explore the models on larger and more diverse mutual fund datasets.

2. Investigate methods to enhance the interpretability of deep learning models.

3. Continuously update and refine the models as new data becomes available.

By automating the analysis of mutual fund prospectuses, this study empowers investors and financial institutions to make more informed and efficient investment decisions.

1. Introduction

Mutual funds are a popular investment vehicle, offering diversified portfolios managed by professionals. However, selecting funds that align with individual investment goals and risk tolerance can be challenging, particularly when manually analyzing complex mutual fund prospectuses. This has led to a growing interest in applying natural language processing (NLP) techniques to automatically extract and predict investment strategies from mutual fund text data.

Early NLP research in finance focused on sentiment analysis and document classification using traditional machine learning methods (Tetlock, 2007; Malo et al., 2014)[1][2]. With the rise of deep learning, researchers began exploring its application to financial text analysis, leveraging convolutional neural networks (CNN) and attention mechanisms to capture semantic features and identify important information (Ding et al., 2015[3]; Liu, 2018)[4]. More recently, transformer-based models like BERT have achieved state-of-the-art performance on various NLP tasks, including financial sentiment analysis (Yang et al., 2016[5]; Araci, 2019[6]).

Building upon these advancements, our research aims to predict mutual fund investment strategies based on their text summaries using NLP techniques. We propose a two-step approach: (1) transforming the text data into meaningful vector representations using the Skip-gram model (Mikolov et al., 2013)[1] and pre-trained BERT embeddings, and (2) training CNN and Recurrent Neural Networks (RNN) to predict investment strategies.

To the best of our knowledge, this is the first study to apply deep learning models for predicting mutual fund investment strategies from text summaries. Our research contributes to the growing literature on the application of NLP in finance and offers practical implications for investors and financial institutions seeking to automate the analysis of mutual fund prospectuses.

The remainder of this paper is organized as follows. Section 2 describes the methodology, Section 3 presents the empirical analysis, and Section 4 concludes the paper and discusses future research directions.

2. Methods

The main objective of our research is to predict a fund's investment strategy based on its summary. This process can be divided into two main parts. The first part, detailed in Section 2.1, involves transforming the text into vectors that can be understood by computers. The second part, described in Section 2.2, focuses on using these vectors to fit a model that predicts the fund's investment strategy.

2.1 Text Vectorization

2.1.1 Skip-gram Model

Skip-gram is a word embedding training method proposed by Mikolov et al. in 2013[7], which is a type of Word2Vec model.

The basic idea of the Skip-gram model is that a word can be used to predict the words in its context. For example, given a sentence "The quick brown fox jumps over the lazy dog", with "fox" as the center word and a skip window size of 2, its context includes "The", "quick", "brown", "jumps", and "over". The goal of the model is to use the word "fox" to predict these context words.

During the training process, the model adjusts the parameters of the word vectors so that the center word's vector can better predict the probability of its context words. The idea behind this approach is the distributional hypothesis, which states that semantically similar words appear in similar contexts. Therefore, the trained word vectors can capture the semantic similarity between words.

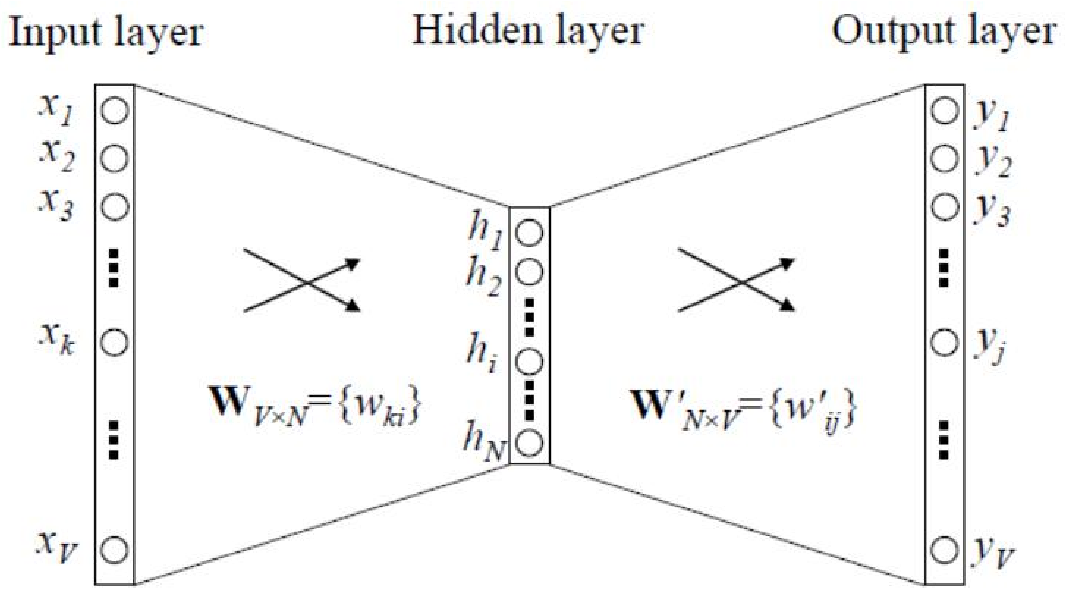


Figure 1: Three-Layer Neural Network for Word Embeddings

Specifically, we build the Skip-gram Model using the following approach: First, we convert the text into a list of words, establish a mapping relationship between words and their indices, and transform the words into one-hot vector representations. Then, we generate the training data by randomly sampling center words and their context words from the text in each epoch and converting them into one-hot vectors. We define a neural network model with an autoencoder structure using Keras. The encoder part maps the one-hot vectors to low-dimensional word vectors, while the decoder part maps the word vectors back to one-hot vectors. The Skip-gram model only uses the encoder part. The model parameters are shown in the table below:

Table 1: Skip-gram Model Hyperparameters

Parameter Name	Value	Description
batch_size	128	Number of samples per batch
num_epochs	2	Number of training epochs
embedding_size	50	Dimension of word vectors, determining their expressive power
max_vocabulary_size	5000	Maximum size of the vocabulary, sorted by word frequency in descending order, taking the top max_vocabulary_size words
min_occurrence	10	Minimum number of occurrences for a word to be included in the vocabulary, used to filter out low-frequency words and reduce noise
skip_window	3	Size of the context window, considering 3 words before and after the center word as its context
num_skips	4	Number of context words sampled for each center word, randomly selecting 4 context words within each window to predict the center word
vocab_size	2999	Actual size of the vocabulary
Total params	302949	Total number of model parameters

We train the model using the autoencoder training approach, aiming to minimize the cross-entropy loss between the decoder's output and the actual one-hot representation of the context words. After the model is trained, the encoder part can map each word to a low-dimensional word vector. Finally, we iterate over all the words to obtain a dictionary mapping words to their corresponding vectors. Each vector in this dictionary stores the word and its corresponding contextual information.

2.1.2 BERT

BERT (Bidirectional Encoder Representations from Transformers)[8] is a pre-trained language representation model that utilizes the encoder structure of the Transformer[9] and is pre-trained on large-scale unlabeled text data. The core idea behind BERT is to learn contextual representations of words by jointly conditioning on both the left and right context. This enables BERT to achieve excellent performance on various NLP tasks.

In our project, we first load the pre-trained BERT model and the corresponding tokenizer. We use the 'distilbert-base-uncased'[10] model, which is a distilled version of BERT. It has a smaller size and faster inference speed while maintaining performance close to the original BERT.

Next, we convert the text into the input format required by BERT. The tokenizer is used to tokenize the text. If the number of tokens after tokenization exceeds BERT's maximum input length (set to 512 in this case), the tokens are truncated. Special `[CLS]` and `[SEP]` tokens are added at the beginning and end of the tokens, respectively. The tokens are then converted to their corresponding IDs. An attention mask is created to indicate which positions of the tokens are actual words and which positions are padding. If the length of the tokens is less than the maximum length, padding is applied using 0.

Finally, the DistilBERT model is used to process each input and extract features.

2.2 Prediction Part

2.2.1 CNN

Convolutional Neural Networks (CNNs) are a type of deep learning algorithm. CNNs have demonstrated outstanding performance in various domains, particularly in image and video analysis, natural language processing, and more. The concept of CNNs can be traced back to the 1980s and was proposed by Yann LeCun[11] and others in the 1990s, initially applied to handwritten digit recognition problems. With the advancement of hardware performance and the availability of big data, CNNs have gained widespread application in various image recognition tasks, especially after the success of AlexNet[12] in the ImageNet[13] competition in 2012, which greatly propelled the research and application of CNNs.

CNNs are typically composed of three types of layers: convolutional layers, pooling layers, and fully connected layers. Convolutional layers are the core of CNNs and are used to extract features from images. By performing sliding window operations with filters (or convolutional kernels) on the input image, multiple feature maps are generated, each representing a specific feature in the image data. Pooling layers are also known as downsampling layers, these layers are used to reduce the spatial size of the feature maps, thereby reducing computational complexity and the risk of overfitting. Common pooling operations include max pooling and average pooling. Fully connected layers located at the end of the CNN, these layers utilize the extracted and compressed feature information from the previous convolutional and pooling layers for the final classification or regression tasks.

We employed two CNN models in our study. The first model, which we refer to as CNN_Small, has fewer layers and a simpler structure. The second model, named CNN_Large, has slightly more layers.

See the appendix for details of the structure

2.2.2 RNN

RNNs (Recurrent Neural Networks) are a widely used neural network architecture for processing sequential data. Unlike traditional feedforward neural networks, RNNs introduce the concept of state, allowing information to persist between time steps of a sequence. This enables

RNNs to capture long-term dependencies in sequential data, achieving significant success in various domains such as natural language processing, speech recognition, and time series prediction.

The concept of RNNs was first proposed by Hopfield[14] in 1982, and further developed by Elman, Jordan, and others in the early 1990s[15]. However, traditional RNNs faced challenges in capturing long-term dependencies due to the problems of vanishing and exploding gradients. To address these issues, researchers proposed improved variants of RNNs, such as Long Short-Term Memory (LSTM)[16] networks and Gated Recurrent Units (GRUs)[17]. These models introduce gating mechanisms and memory cells, effectively mitigating the gradient problems and enabling RNNs to learn longer sequence dependencies.

3 Empirical Analysis

3.1 Data

The primary objective of this study is to predict the investment strategy employed by each mutual fund using their text summaries.

This research will utilize two publicly available text corpora related to mutual funds, primarily sourced from mutual fund prospectuses. The first corpus, MutualFundSummary, contains approximately 500 mutual fund summaries stored as text files. Each summary encapsulates the fund's objectives, principal strategies, fee breakdown, major risks, and other pertinent information regarding fund management, forming a crucial part of the prospectus and providing key insights into the fund. The second corpus, MutualFundLabels.csv, is a table containing information on 467 mutual funds, including: id (unique identifier for each fund), fund_name (name of the fund, corresponding to the file names in MutualFundSummary), Performance fee? (indicating whether the fund charges a performance fee), Investment Strategy (the investment strategy adopted by the fund), Leverage? (indicating whether the fund employs leverage), Portfolio Composition (the composition of the fund's investment portfolio), and Concentration (the degree of concentration of the fund).

Upon examining the data, it was observed that the "Performance fee?" column in MutualFundLabels had 447 missing values, leading to its removal. Furthermore, the Investment Strategy feature revealed that the Long Short Funds and Commodities Fund categories had very few samples, with only 4 and 1 instances respectively, constituting a small fraction of the total investment strategy categories. Including these categories could interfere with the model's learning process and deviate from the primary objective of training the model. Consequently, these two investment strategies were excluded, resulting in a remaining 462 observations.

The fund_name column contains the names of the mutual funds, allowing for a one-to-one matching with the summaries. The txt format summaries and the corresponding labels from the table were matched using the fund_name column.

The original data comprised 545 summaries, of which 84 summaries lacked corresponding labels. These label-less summaries were set aside as the test set. The remaining summaries were combined with the processed MutualFundLabels.csv data to form the initial dataset. The remaining 461 observations were then divided into a training set (368 observations) and a validation set (93 observations) using an 8:2 ratio.

Data cleaning was performed to prepare the text input for prediction. This involved converting complete sentences into individual words and removing meaningless stop words. Specifically, newline characters (\n) and tab characters (\t) were replaced with spaces, and the text was converted to lowercase. The word_tokenize function from the nltk library was used to tokenize the text, splitting it into a list of words. Stop words, which are commonly ignored words in text processing such as "the," "a," and "an," were filtered out. Non-alphabetic characters were also removed using regular expressions, resulting in a filtered list of words called filtered_sentence.

The Skip-gram Model was employed for word vectorization, with a detailed explanation provided in section 2.1.1.

Lemmatization, the process of reducing a word to its base form, was performed using the WordNetLemmatizer from the NLTK natural language processing library. By lemmatizing the words, the vocabulary diversity was reduced, enhancing the efficiency and accuracy of text processing.

A knowledge base was then created by extracting keywords using TF-IDF. The TF-IDF formula is given by TF-IDF = TF * IDF, where TF represents the frequency of a word in a document, and IDF = log(total number of documents / number of documents containing the word). Higher TF-IDF values indicate greater importance of the word to the document. Pre-trained word embeddings were used to find the most similar words to each keyword, and the keywords and similar words were combined to construct an expanded knowledge base. The relevance of sentences to the knowledge base was computed by averaging the word vectors in each sentence to obtain a sentence centroid vector, calculating the cosine distance between the sentence centroid and each keyword in the knowledge base, and taking the average of the n_closer smallest distances as the relevance score for the sentence. The num_sent sentences with the lowest scores were returned as the most relevant sentences to the knowledge base. This step aimed to reduce noise interference and identify sentences most relevant to the Investment Strategy, thereby reducing the amount of data the model needed to process and accelerating the training speed.

Through these data processing steps, the raw text summaries were transformed into a format suitable for predicting investment strategies, and a knowledge base was constructed to compute sentence relevance.

Knowledge Base = {'shareholder', 'sells', 'ask', 'share', 'performance', 'fund', 'years', 'achieve', 'fiscal', 'shows', 'return', 'company', 'marginal', 'december', 'fixed-income', 'seeks', 'cause', '12b-1', 'reimbursement', 'portfolio', 'waivers', 'end', 'economic', 'n', 'turnover', 'rate', 'year', 'inception', 'depend', 'cash', 'international', 'foreign', 'turns', 'class', 'result', 'differently', 'debt', 'prices', 'affected', 'generally', 'including', 'assumes', 'recent', 'pay', 'pays', 'market', 'credit', 'c', 'expense', 'exempt', 'institutional', 'objective', 'risk', 'distributions', 'value', 'tax', 'losses', 'indicate', 'security', 'ended', 'r', 'waiver', 'b', 'relative', 'fee', 'extent', 'periods', 'could', 'may', 'classes', 'investment', 'conditions', 'r6', 'personal', 'reimbursements', 'decline', 'past'}

The method described above was used to select the key sentences from the training set, validation set, and test set.

Another approach to data processing involves directly vectorizing the summary text using a pre-trained BERT model. This method converts the text into vectors that can be fed into a neural network. The specific details of this approach are outlined in section 2.1.2.

By employing these two techniques – selecting relevant sentences using the knowledge base and vectorizing the text using a pre-trained BERT model – the text summaries are transformed into a format suitable for training and evaluating machine learning models to predict the investment strategies of mutual funds.

3.2 Results

In the text vectorization section, both self-trained Word2Vec models and pre-trained BERT models were employed. During the Word2Vec vectorization process, a step involving the use of a knowledge base for key sentence selection was included, which we refer to as Feature Selection (FS).

Throughout the training process, the ModelCheckpoint callback function was utilized to save the best-performing model on the validation set after each epoch for every model. This model was then used as the final model for prediction.

The loss function selected was categorical_crossentropy, a common choice for multi-class classification problems. The training platform used was Google's online Colab, with the GPU set to T4. The evaluation metric determined by the optimizer during training was Accuracy.

The adjustments made in CNN_Large compared to CNN_Small were twofold: firstly, the model size was increased, and secondly, class_weights were used to balance the issue of varying class sizes within the dataset. The class_weights values were set as the reciprocal of the proportion of each class in the validation set.

Table 2: Model Parameters and Performance

Text Vectorization	FS	FS	FS	BERT	BERT	BERT
model	CNN_Small	CNN_Large	RNN	CNN_Small	CNN_Large	RNN
epoch	800	800	800	800	800	800
batch_size	128	128	128	128	128	128
learning_rate	0.001	0.001	0.001	0.001	0.001	0.001
Training time	105.33 s	156.76 s	76.36 s	56.26 s	105.62 s	105.62s
Optimizer	RMSprop	RMSprop	RMSprop	Adam	Adam	Adam
Total params	147531	527653	266853	129955	1143235	38949891
F1-score (Average)	0.81	0.80	0.83	0.83	0.85	0.84
AUC	0.8356	0.8311	0.8712	0.8446	0.8568	0.8490
Accuracy Score	0.8064	0.7957	0.8279	0.8279	0.8495	0.8387

Table 3 compares the predictive performance of different Text Vectorization Methods and neural network models on the Validation Set, along with a portion of the parameter settings. To facilitate the comparison of the impact of models and Text Vectorization Methods on training results, we specifically controlled the epoch, batch_size, and learning_rate to be the same. For the data that underwent Feature Selection, it can be observed that the RNN model performed the best, with the best performance in training time, F1-score, AUC, and Accuracy Score. Comparing CNN_Small and CNN_Large, although the number of parameters increased by three times and the training time also increased, the performance of CNN_Large did not improve significantly. They obtained very similar AUC and Accuracy Scores, and even slightly lower values. We speculate that the reason may be that a slightly larger model may have overfitted the training set, which in turn affected its performance on the validation set.

When using pre-trained BERT for text vectorization and then feeding it into the model, it was found that the prediction accuracy improved significantly. The main reason is that this data has not been truncated, retaining more information compared to the data that underwent Feature Selection. In this case, models with a larger number of parameters perform much better than models with a smaller number of parameters, as there is sufficient information for the model to learn. Therefore, comparing all the models, the model trained on CNN_Large using BERT for word vector extraction performed the best, with an F1-score (Average) of 0.85, AUC of 0.8568, and Accuracy Score of 0.8495 on the validation set. We used this model to predict the investment strategies of funds in the test set, and the prediction results are stored in y_test_CNN.csv. Figure 2 shows the ROC Curve of BERT_CNN_Large, which demonstrates a good fit.

Additionally, I also attempted to directly use the pre-trained BERT model to train the training set as a downstream task and then predict on the validation set, but the model's performance was not satisfactory. On one hand, the original BERT has a very large number of parameters, and even if only a portion of the parameters is updated downstream, the time taken for one epoch is similar to the time taken for 800 iterations of training on our custom model. When updating parameters on BERT, a rapid increase in accuracy can be observed, but it enters a plateau period after reaching around 60% and cannot achieve better performance. We speculate that the reason is that the original BERT is designed for general text rather than financial text, so the prediction on this fund's summary has an initial improvement but cannot be sustained.

It is worth noting that during multiple training processes, the results obtained by the model are not entirely consistent each time, and the performance on some benchmarks may also have slight differences. However, overall, the results will stabilize within a range with little variation.

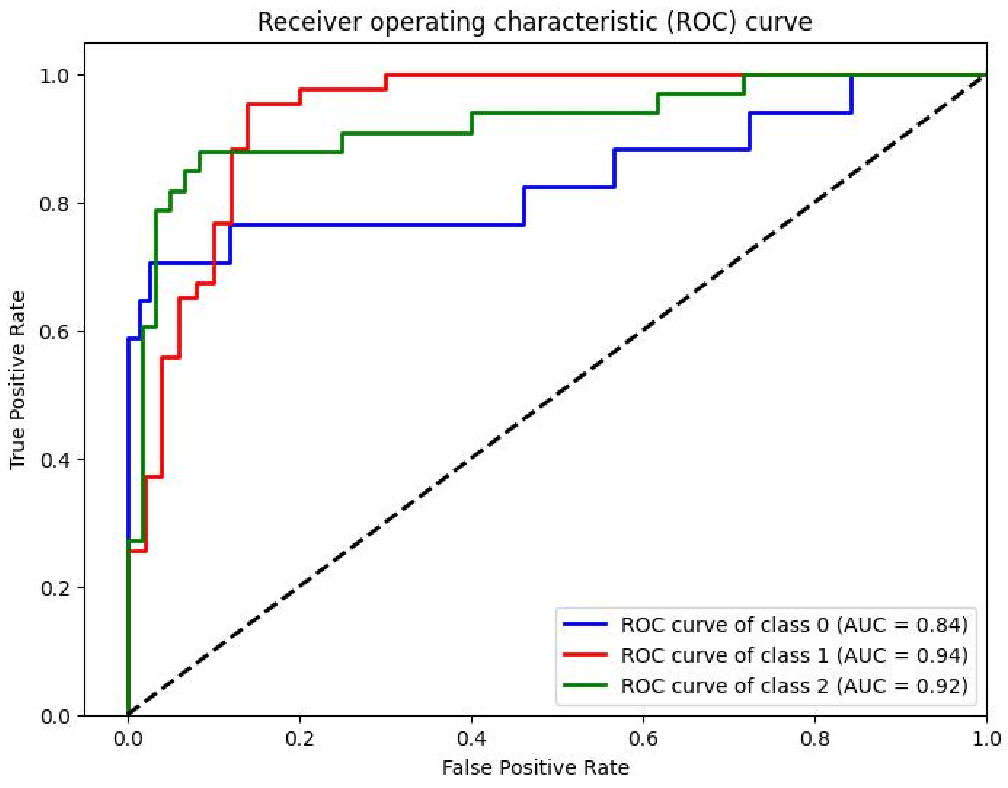


Figure 2: ROC Curve of BERT_CNN_Large

4 Conclusions and Discussions

This study explores the application of deep learning models for predicting mutual fund investment strategies from text summaries. By employing text vectorization techniques, including self-trained Word2Vec models with feature selection and pre-trained BERT embeddings, followed by training CNN and RNN models, we demonstrate the feasibility and effectiveness of this approach.

The empirical analysis reveals several key findings. Firstly, for data that underwent feature selection using a knowledge base, the RNN model exhibited the best performance in terms of training time, F1-score, AUC, and accuracy score. Secondly, when comparing CNN_Small and CNN_Large, the larger model did not significantly improve performance, potentially due to overfitting on the training set. Thirdly, using pre-trained BERT embeddings for text vectorization led to a substantial improvement in prediction accuracy, with the CNN_Large model achieving the highest performance metrics on the validation set (F1-score: 0.85, AUC: 0.8568, accuracy score: 0.8495).

However, the study has certain limitations. Firstly, the dataset used in this research is relatively small, which may limit the generalizability of the findings. Future studies should explore the application of these models on larger and more diverse datasets to validate their effectiveness. Secondly, the interpretability of deep learning models remains a challenge, making it difficult to understand the underlying reasoning behind the predictions. Developing methods to enhance the interpretability of these models is crucial for building trust and facilitating their adoption in real-world financial decision-making.

Contribution

The work is equally done by two group members, with the data preprocessing being a collaborative effort. Subsequently, Jiayang Xu undertook the training of Skip-gram and CNN models, while Ziyuan Zhang handled the training of BERT and RNN models, and correspondingly drafted the respective report sections.

Reference

[1] Tetlock, Paul C. "Giving content to investor sentiment: The role of media in the stock market." The Journal of finance 62, no. 3 (2007): 1139-1168.

[2] Malo, Pekka, Ankur Sinha, Pekka Korhonen, Jyrki Wallenius, and Pyry Takala. "Good debt or bad debt: Detecting semantic orientations in economic texts." Journal of the Association for Information Science and Technology 65, no. 4 (2014): 782-796.

[3] Ding, Xiao, Yue Zhang, Ting Liu, and Junwen Duan. "Deep learning for event-driven stock prediction." In Twenty-fourth international joint conference on artificial intelligence. 2015.

[4] Liu, Huicheng. "Leveraging financial news for stock trend prediction with attention-based recurrent neural network." arXiv preprint arXiv:1811.06173 (2018).

[5] Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. "Hierarchical attention networks for document classification." In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, pp. 1480-1489. 2016.

[6] Araci, Dogu. "Finbert: Financial sentiment analysis with pre-trained language models." arXiv preprint arXiv:1908.10063 (2019).

[7] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems 26 (2013).

[8] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[9] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[10] Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." arXiv preprint arXiv:1910.01108 (2019).

[11] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.

[12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." Communications of the ACM 60, no. 6 (2017): 84-90.

[13] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In 2009 IEEE conference on computer vision and pattern recognition, pp. 248-255. Ieee, 2009.

[14] Hopfield, John J. "Neural networks and physical systems with emergent collective computational abilities." Proceedings of the national academy of sciences 79, no. 8 (1982): 2554-2558.

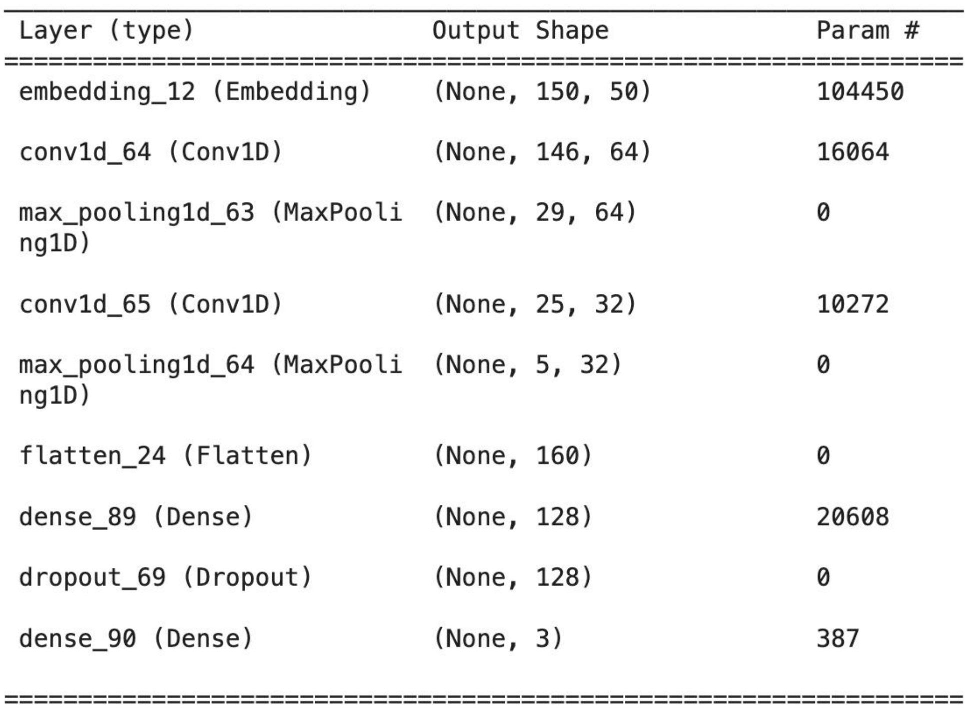
[15] Elman, Jeffrey L. "Finding structure in time." Cognitive science 14, no. 2 (1990): 179-211.

[16] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

[17] Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

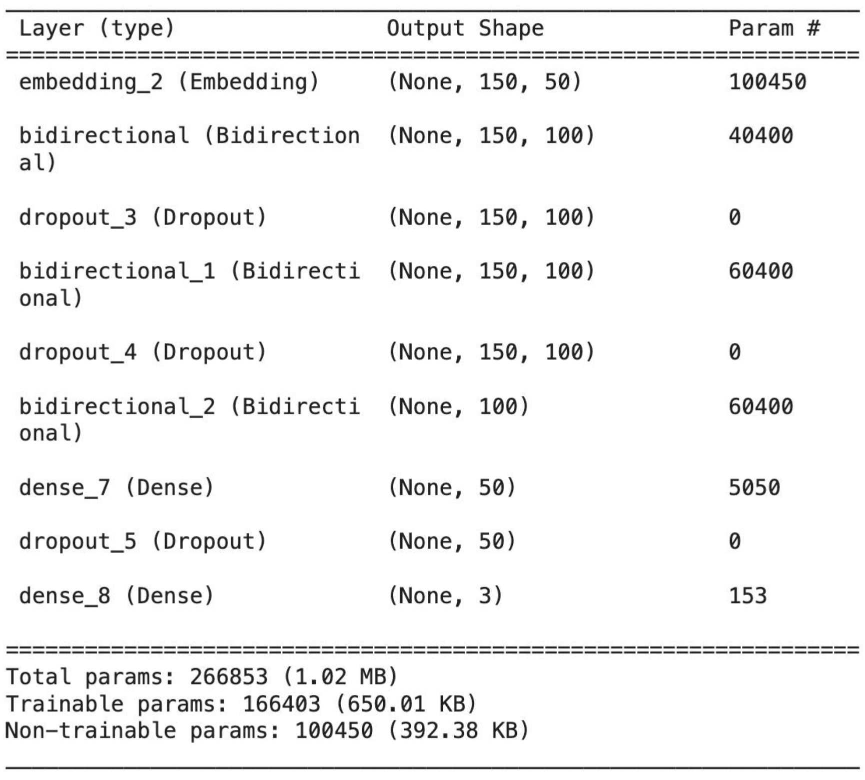
Appendix

Model Structure of CNN_Small



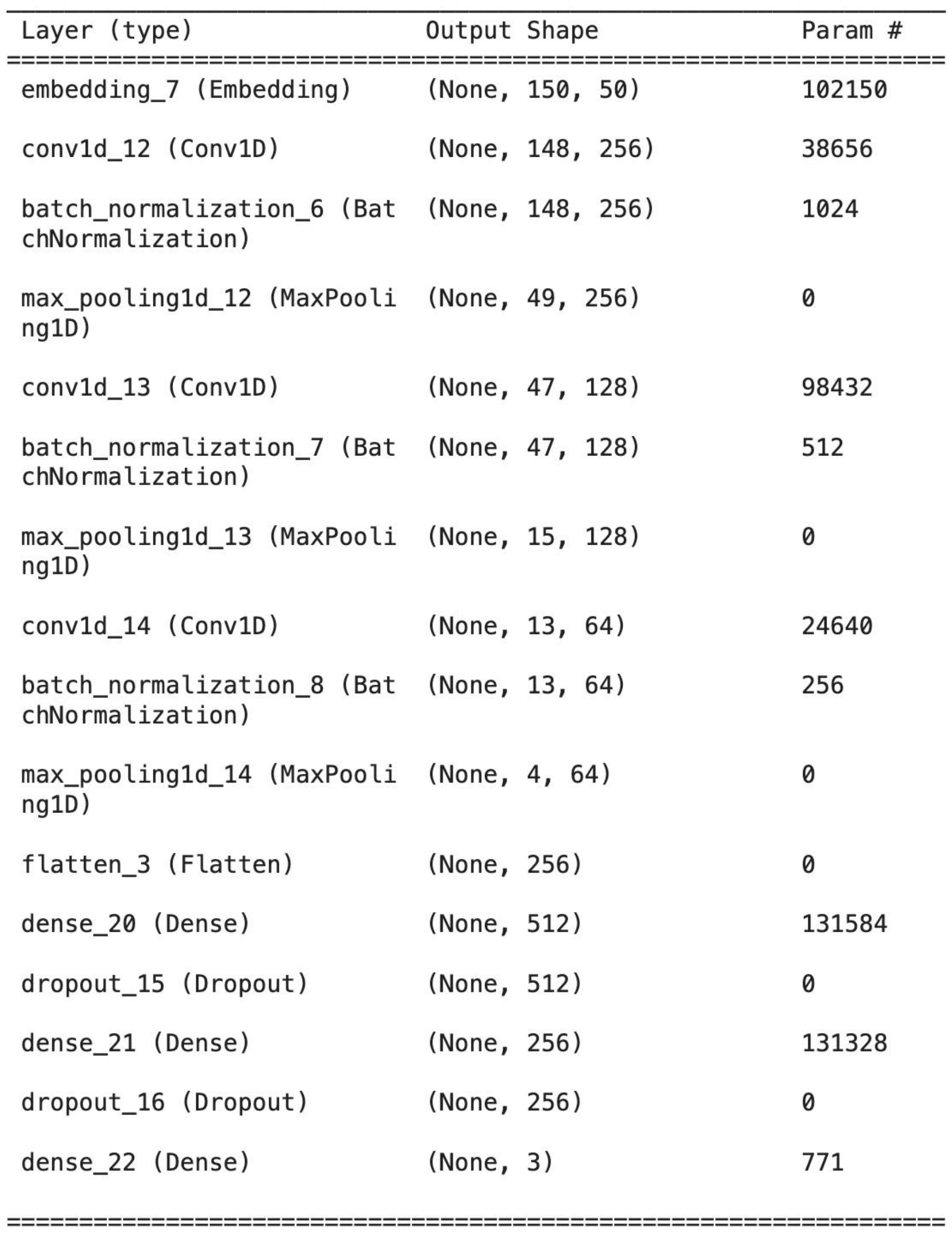
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_12 (Embedding)    (None, 150, 50)           104450

 conv1d_64 (Conv1D)          (None, 146, 64)           16064

 max_pooling1d_63 (MaxPooli  (None, 29, 64)            0
 ng1D)

 conv1d_65 (Conv1D)          (None, 25, 32)            10272

 max_pooling1d_64 (MaxPooli  (None, 5, 32)             0
 ng1D)

 flatten_24 (Flatten)        (None, 160)               0

 dense_89 (Dense)            (None, 128)               20608

 dropout_69 (Dropout)        (None, 128)               0

 dense_90 (Dense)            (None, 3)                 387

=================================================================
```

Model Structure of RNN



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 150, 50)           100450

 bidirectional (Bidirection  (None, 150, 100)          40400
 al)

 dropout_3 (Dropout)         (None, 150, 100)          0

 bidirectional_1 (Bidirecti  (None, 150, 100)          60400
 onal)

 dropout_4 (Dropout)         (None, 150, 100)          0

 bidirectional_2 (Bidirecti  (None, 100)               60400
 onal)

 dense_7 (Dense)             (None, 50)                5050

 dropout_5 (Dropout)         (None, 50)                0

 dense_8 (Dense)             (None, 3)                 153

=================================================================
Total params: 266853 (1.02 MB)
Trainable params: 166403 (650.01 KB)
Non-trainable params: 100450 (392.38 KB)
_________________________________________________________________
```

Model Structure of CNN_Large



```
________________________________________________________________
 Layer (type)                Output Shape              Param #
================================================================
 embedding_7 (Embedding)     (None, 150, 50)           102150

 conv1d_12 (Conv1D)          (None, 148, 256)          38656

 batch_normalization_6 (Bat  (None, 148, 256)          1024
 chNormalization)

 max_pooling1d_12 (MaxPooli  (None, 49, 256)           0
 ng1D)

 conv1d_13 (Conv1D)          (None, 47, 128)           98432

 batch_normalization_7 (Bat  (None, 47, 128)           512
 chNormalization)

 max_pooling1d_13 (MaxPooli  (None, 15, 128)           0
 ng1D)

 conv1d_14 (Conv1D)          (None, 13, 64)            24640

 batch_normalization_8 (Bat  (None, 13, 64)            256
 chNormalization)

 max_pooling1d_14 (MaxPooli  (None, 4, 64)             0
 ng1D)

 flatten_3 (Flatten)         (None, 256)               0

 dense_20 (Dense)            (None, 512)               131584

 dropout_15 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 256)               131328

 dropout_16 (Dropout)        (None, 256)               0

 dense_22 (Dense)            (None, 3)                 771

================================================================
```