Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Data Science

Quantum investment strategies: LSTM based Sentiment Analysis for portfolio prediction

Relatore: Prof. Simone Bianco

Tesi di Laurea Magistrale di:

Claudio Maffi

Matricola 875789

Anno Accademico 2021-2022

Contents

Introduction

Quantum investing is a relatively new field that involves the use of quantum computers and algorithms to analyse and make investment decisions. Investment strategies, for a long time, have been a subject of interest for both academic researchers and practitioners. One typical approach in the investment strategies is the use of neural networks, which are artificial intelligence systems that are inspired by the structure and function of the human brain. In the most recent years, there has been registered a growing quantity of research around the use of neural networks for investment strategies, with a particular focus on long short-term memory (LSTM) networks. Long short-term memory models are a type of recurrent neural network that have been widely used because they are capable of learning long-term dependencies in sequential data, making them well-suited for tasks that require understanding of context and temporal dynamics. Moreover, LSTM models have been shown to perform well on a variety of NLP tasks, including language translation, machine translation, and language modelling. Sentiment analysis, also known as opinion mining, involves using computational methods to analyze and extract subjective information from text data (Bing, 2012). The attention around sentiment analysis has significantly increased in the past few years due to its potential to provide valuable insights in various areas, such as marketing, customer service and politics. There are several approaches to sentiment analysis, in specific some of them are: lexicon-based, rule-based, and machine learning-based. Lexicon-based approaches rely on the use of pre-defined dictionaries or lexicons of words associated with specific sentiments (e.g., positive or negative) to classify the sentiment of text. Rule-based approaches involve the use of predefined rules or heuristics to classify the sentiment of text. Machine learning-based approaches involve the use of machine learning algorithms, such as support vector machines, decision trees and neural networks, to learn patterns in the data and classify the sentiment of text. In addition, by simply analysing historical financial data, sentiment analysis has also been identified as a potentially useful tool for investment strategies. One acknowledged tool for sentiment analysis is BERT (Bidirectional Encoder Representations from Transformers), which is a state-of-the-art NLP model developed by Google (Devlin, Chang, Lee, & Toutanova, 2018). It is structured on the transformer architecture, which can effectively model long-range dependencies in text data (Vaswani, et al., 2017). BERT has been tested to outperform previous state-of-the-art models on a variety of NLP tasks, including language modelling, machine translation and natural language understanding. It has also been used for sentiment analysis tasks, such as sentiment classification, aspect-based sentiment analysis and stock market sentiment analysis. However, to the best of our knowledge, the use of BERT for sentiment analysis in the context of quantum investment strategies has not yet been explored in depth. In this study, we aim to evaluate the performance of an LSTM model from the Qlib repository (Yang, Liu, Zhou, Bian, & Liu, 2020) developed by Microsoft and a modified LSTM model that includes a sentiment analysis feature calculated with BERT on tweets related to general news. We aim to know if the sentiment feature can improve the LSTM portfolio performance following a specific investment strategy. We will compare the performance of the two models by taking into consideration annual return and risk of an investment portfolio.

In the first chapter we will provide an overview of the Qlib repository, including its history, key features, and status. We will also introduce the concept of active and passive investment strategies, which are methods used to make decisions about how to allocate resources to achieve a desired return on investment. Consequently, we will discuss the concept of sentiment analysis with BERT, and we will report examples of previous research on combining sentiment analysis with LSTM models. In the second chapter we will demonstrate the retrieving and cleaning process of historical financial data and tweets data. Lastly, we will explain the implementation of LSTM models with and without sentiment feature. In the third chapter we will analyse the results and we will compare the metrics and models' performance. In the conclusion chapter, we will present the main findings of our study and discuss their implications for quantum investment field. Finally, we will discuss any limitations of the study and suggest directions for future research.

Literature review

2.1 Overview of Qlib repository

The Qlib repository was first launched in 2018 as a joint project between researchers at the University of Tokyo and the National Institute of Informatics. Since then, it has grown to become a widely used resource for quantum computing research, with a growing community of contributors from around the world. It aims to provide a resource for those interested in exploring the potential of quantum computing, as well as a platform for developing and testing new algorithms and strategies.

Qlib is a repository of tools and algorithms for quantum computing applications, including optimization, simulation, and machine learning. The repository is open source and maintained by a community of researchers and developers, who contribute and review code to ensure its quality and reliability. It includes a user-friendly interface with a range of tutorials and documentation, as well as a comprehensive set of examples to help users get started. Qlib has an active community of contributors from around the world and is designed to be compatible with a range of quantum computing platforms. It supports different programming languages, including Python, C++, and Julia, and is used by educators around the world to explore the potential of quantum computing and to teach others about the field. It includes a range of resources and materials, such as papers and lectures to support these efforts.

In figure 1 below, the Qlib's structure is represented. As we can see, it is defined by the following components:

- Infrastructure Layer: it provides underlying support for Quant research. DataServer module provides high-performance infrastructure for users to manage and retrieve raw data, while Trainer module provides flexible interface to control the training process of models which enable algorithms controlling the training process.
- Learning Framework Layer: The Forecast Model and Trading Agent are learnable. They are learned based on the Learning Framework Layer and then applied to multiple scenarios in Workflow Layer. The supported learning paradigms can be categorized into Reinforcement Learning and Supervised Learning. The Learning Framework leverages the Workflow Layer as well, for example sharing Information Extractor or creating environments based on Execution Environment.
- Workflow Layer: the Workflow Layer is the core of the process and it covers the whole workflow of quantitative investment. Here both Supervised-Learning-based strategies and RL-based Strategies are supported, and the workflow changes depending on which strategy has been chosen. Information Extractor module extracts data for models, while Forecast Model module focuses on producing all kinds of forecast signals (for example alpha, risk) for other modules. With these signals Decision Generator module will generate the target trading decisions (i.e., portfolio, orders). If RL-based Strategies are adopted, the Policy is learned in an end-to-end way and the trading decisions are generated directly. Decisions will be executed by Execution Environment (i.e., the trading market);
- Interface Layer: Interface Layer tries to present a user-friendly interface for the underlying system. Analyser module will provide users detailed analysis reports of forecasting signals, portfolios, and execution results.

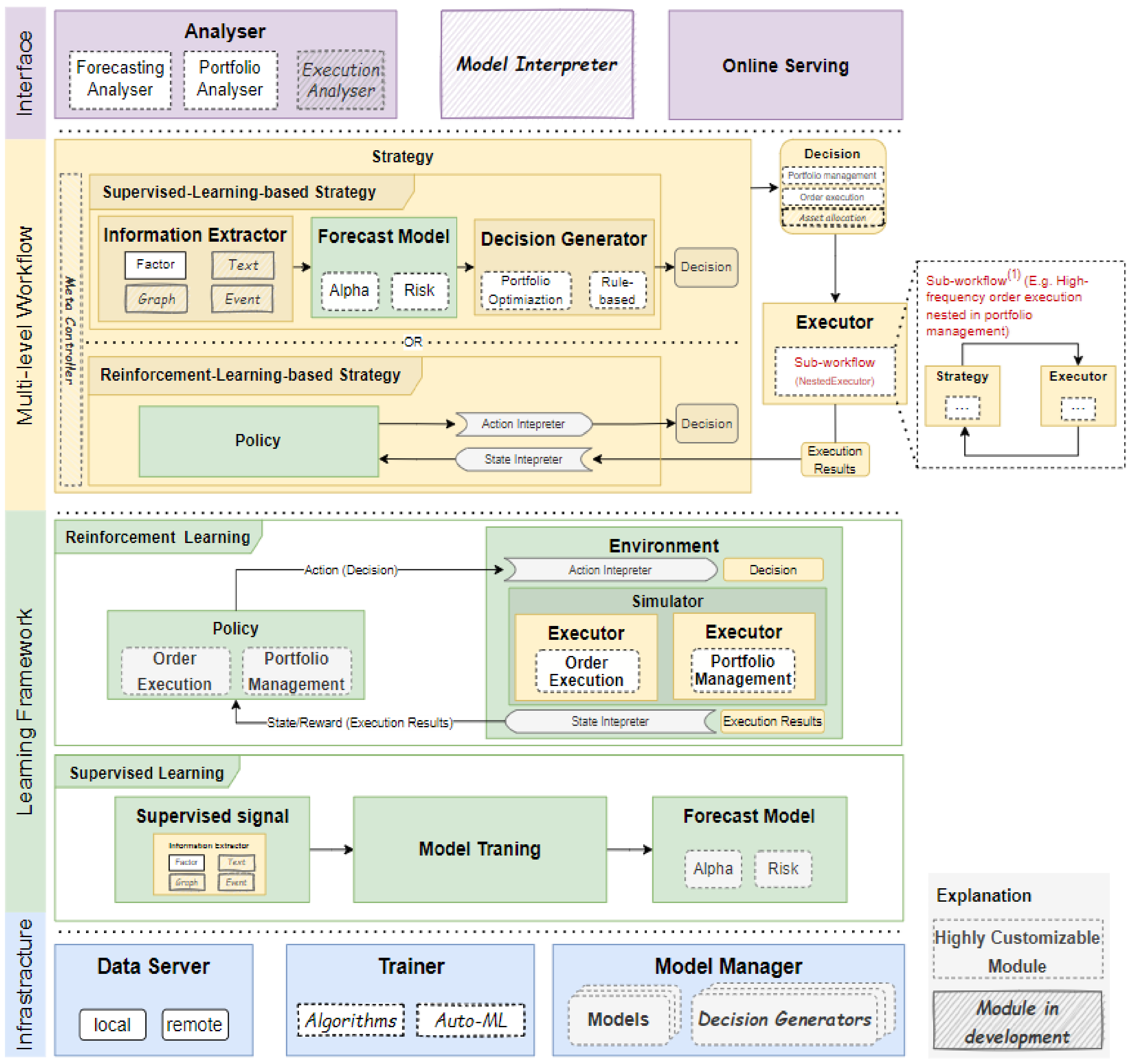


Figure 1: Framework of Qlib repository

Reference: Qlib documentation (Microsoft, Introduction, 2020)

2.2 Investment strategies: difference between active and passive strategies

An investment strategy is a plan or set of guidelines used to make decisions about how to allocate resources to achieve a desired return on investment. Investment strategies can be applied to a wide range of financial assets, including for example stocks, bonds, and real estate. There are many different types of investment strategies, each with its own set of goals and assumptions.

Investments strategies are divided between active and passive: active investment strategies involve actively selecting and managing a portfolio of assets based on certain criteria. Active strategies typically involve higher fees and higher levels of portfolio turnover, as they require ongoing research and analysis to make investment decisions. Passive investment strategies, on the other hand, involve tracking the performance of a specific market index, such as the S&P 500 or CSI 300, and attempting to match the returns of the market. Passive strategies generally involve low fees and low levels of portfolio turnover, as they do not require active management of individual assets.

Some common investment strategies include value investing, growth investing, and index investing. Value investing, for example, is a strategy that focuses on finding undervalued assets that have the potential to appreciate over time. Growth investing, on the other hand, focuses on investing in companies that are expected to grow rapidly in the future. Instead index investing is a pure passive investment strategy that involves tracking the performance of a specific market index. This approach aims to achieve the returns of the market as a whole, rather than trying to outperform it through active management.

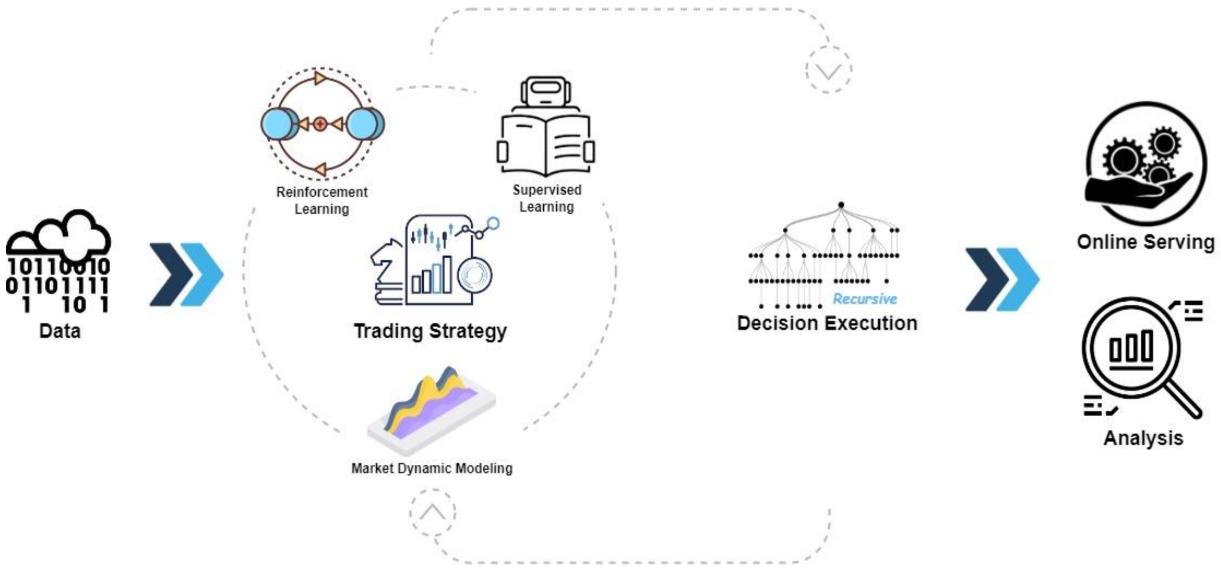


Figure 2: Framework abstract of QLib

Reference: Qlib Github (Microsoft, Github, 2020)

As we can see in Figure 2, the investment strategy is the core of the process. In an effort to yield the LSTM model built with Qlib and the LSTM model tested on data with the average daily sentiment comparable, a common investment strategy has been chosen: the top-k dropout investment strategy.

The top-k dropout investment strategy is a recently developed approach that combines elements of both value and growth investing. As shown in Figure 3, it involves identifying a list of the top k assets that are expected to have the highest returns, and then "dropping out" any assets that do not meet certain performance criteria. This strategy aims to identify the best investment opportunities while also limiting risk by avoiding assets that are not performing well. It has been shown to be effective in certain market conditions, but like any investment strategy, it carries its own set of risks and uncertainties.

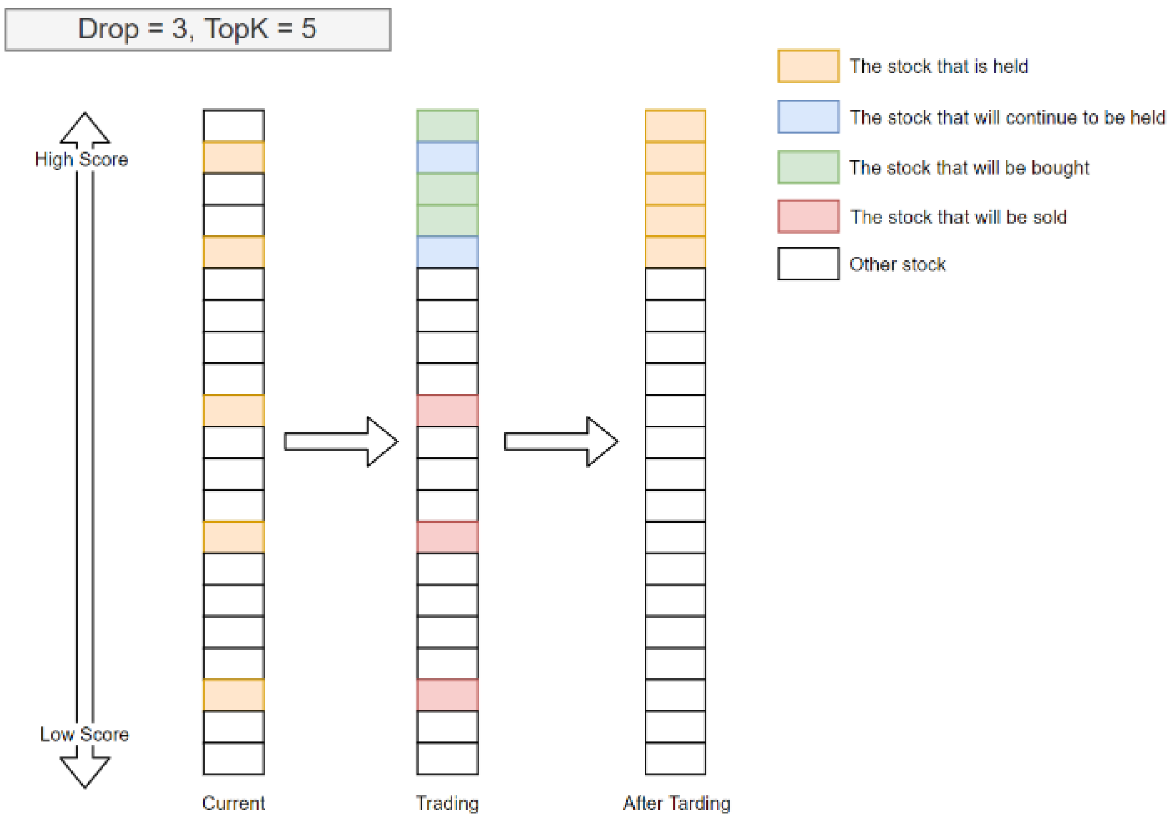


Figure 3: Top-k dropout strategy example with drop=3 and top k=5

Reference: Qlib documentation (Microsoft, Strategy, 2020)

2.3 Recurrent neural networks

Recurrent neural networks (RNNs) are a type of neural network that are designed to process sequential data, such as time series, natural language, and audio signals. They are called "recurrent" because they perform the same computations for every time step in the input sequence, with the output at each time step being fed back into the network as input for the next time step.

RNNs are composed of units called "neurons," which are like the neurons in other types of neural networks. Each neuron receives input from other neurons or from the input data and produces an output that is passed on to other neurons or to the output layer of the network. In an RNN, the inputs and outputs of each neuron are also passed through a set of weights and biases, which are used to adjust the strength of the connections between neurons.

One of the key features of RNNs is that they contain hidden states that are updated at each time step based on the input at that time step and the previous hidden state. This allows RNNs to effectively capture the dependencies between successive time steps, and to make predictions based on this information.

RNNs can be trained using a variant of backpropagation known as "backpropagation through time," which allows the gradients to be propagated back through the sequence of hidden states to update the model parameters. Backpropagation is a method of training artificial neural networks by adjusting the weights and biases of the network based on the error between the predicted output and the true output. It is called "backpropagation" because it involves propagating the error backwards through the network, starting from the output layer and working backwards through the hidden layers. The key idea behind backpropagation is to use the gradient of the error with respect to the model parameters to update the parameters in a way that reduces the error. This is done by calculating the gradient of the error with respect to the output of each neuron, and then propagating this gradient backwards through the network to calculate the gradients with respect to the weights and biases at each layer. The gradients are then used to update the weights and biases using an optimization algorithm, such as stochastic gradient descent. However, it can be computationally intensive, especially for large and complex networks.

As we can see in Figure 4, there are several different types of RNNs, including simple RNNs, gated recurrent units (GRUs), and long short-term memory (LSTM) networks. Each type of RNN has its own set of strengths and weaknesses, and each model is better suited for certain types of tasks. For example, LSTMs are particularly well-suited for modelling long-term dependencies in sequential data, while GRUs are simpler and faster to train but may be less effective at capturing long-term dependencies. In this study, we decided to implement a long short-term memory recurrent neural network.

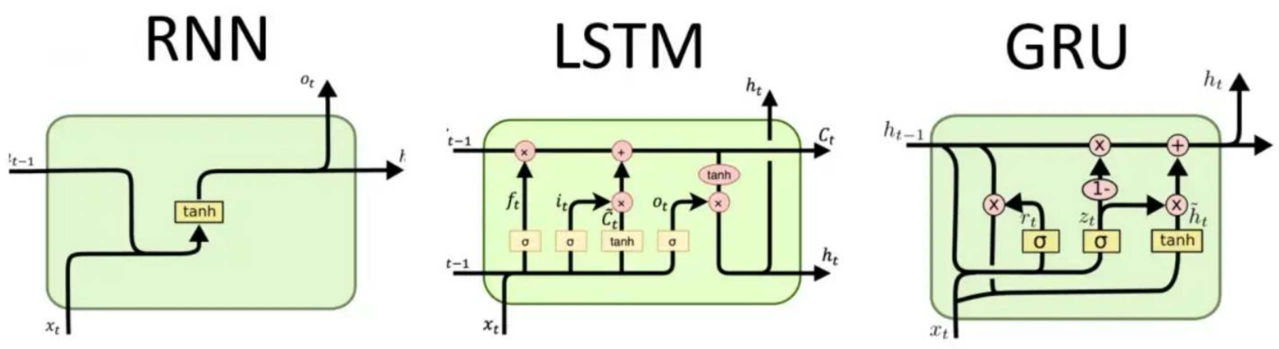


Figure 4: Simple RNN, LSTM and GRU neural networks

Reference: Medium (Medium, 2020)

2.4 Long short-term memory neural network

Traditional RNNs suffer from a problem known as the vanishing gradient problem, which arises when the gradient of the error with respect to the model parameters becomes very small over the course of training. This can make it difficult for the model to learn long-term dependencies, as the information about the earlier time steps becomes diluted as it is passed through the hidden state.

To address this problem, a team of researchers at the University of Toronto, including Sepp Hochreiter and Jürgen Schmidhuber (Hochreiter & Schmidhuber, 1997), developed the LSTM model which uses a set of special "memory cells" to store and manipulate information over long periods of time. The LSTM model includes a set of input, output, and forget gates that control the flow of information into and out of the memory cells.

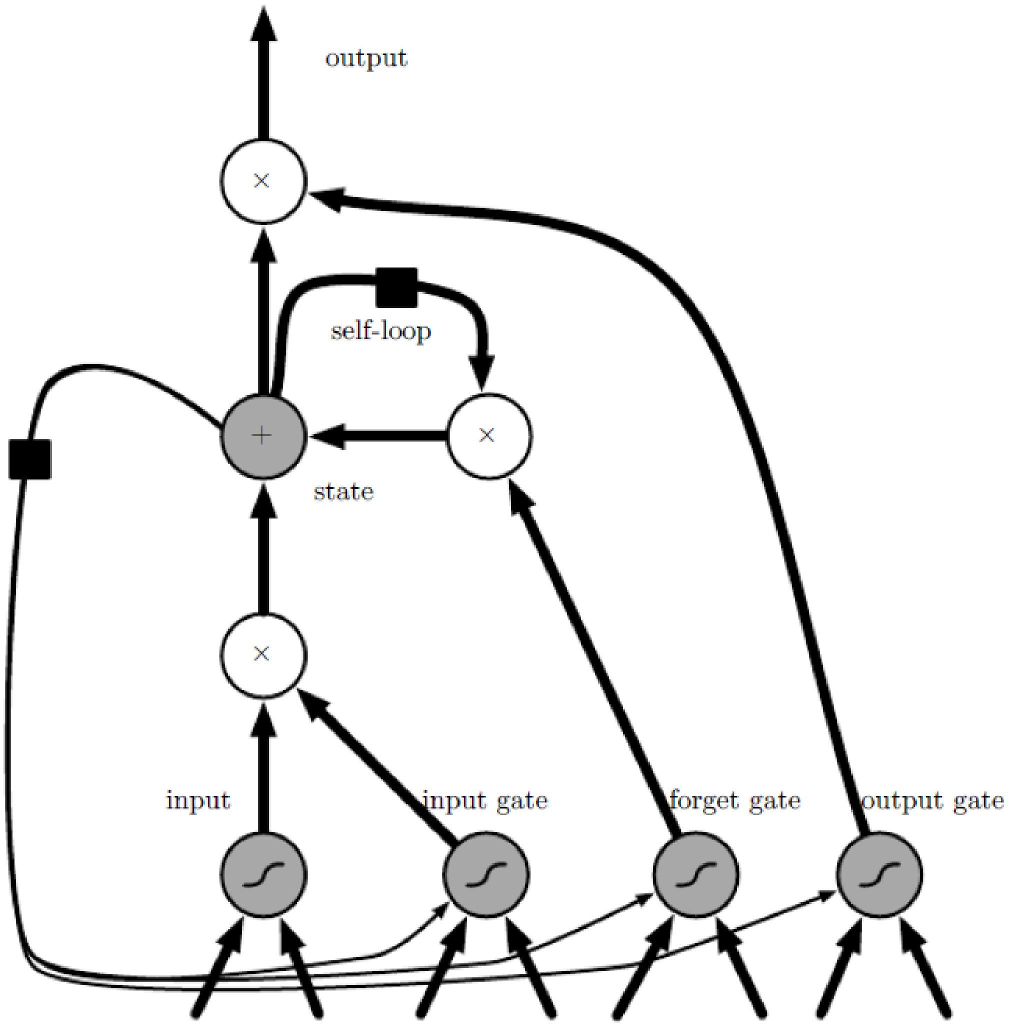


Figure 5: Example of LSTM structure

Reference: Deep Learning (Goodfellow, Bengio, & Courville, 2016)

In Figure 5 we can see more in detail the LSTM recurrent neural network "cell". LSTM process can be resumed in these steps:

- Input: The LSTM takes in a sequence of input data, such as a time series or a natural language text and processes it one time step at a time. At each time step, the input data is passed through a set of weights and biases to produce a set of activations that are used to update the hidden state.

- Hidden state: The hidden state is a set of values that are updated at each time step based on the input data and the previous hidden state. The hidden state is used to capture the dependencies between successive time steps and to make predictions based on this information.
- Input, output, and forget gates: The LSTM uses a set of input, output, and forget gates to control the flow of information into and out of the memory cells. The input gate determines which information from the input and hidden state to store in the memory cells, the output gate determines which information from the memory cells to output, and the forget gate determines which information to discard.
- Memory cells: The memory cells are a set of values that are used to store and manipulate information over long periods of time. They are updated at each time step based on the input, output, and forget gates, and are used to store information that is relevant for making predictions.
- Output: The output of the LSTM is computed at each time step based on the hidden state and the memory cells. The output is passed through a set of weights and biases to produce the final prediction (Goodfellow, Bengio, & Courville, 2016).

Mathematically, the forget gate $f_i^{(t)}$ (for time step *t* and cell *i*) is calculated as follow:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right),$$

Reference: Deep Learning (Goodfellow, Bengio, & Courville, 2016) where $x^{(t)}$ is the current input vector and $h^{(t)}$ is the current hidden layer vector, containing the outputs of all the LSTM cells, and b^f, U^f, W^f are respectively biases, input weights and recurrent weights for the forget gates.

The LSTM cell internal state $s_i^{(t)}$ is thus updated as follows, but with a conditional self-loop weight $f_i^{(t)}$:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right),$$

Reference: Deep Learning (Goodfellow, Bengio, & Courville, 2016)

where *b, U* and *W* respectively denote the biases, input weights and recurrent weights into the LSTM cell. The external input gate unit $g_i^{(t)}$ is computed similarly to the forget gate (with a sigmoid unit to obtain a gating value between 0 and 1), but with its own parameters:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right).$$

Reference: Deep Learning (Goodfellow, Bengio, & Courville, 2016)

The output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the output gate $q_i^{(t)}$, which also uses a sigmoid unit for gating:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$
$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right)$$

Reference: Deep Learning (Goodfellow, Bengio, & Courville, 2016) which has parameters b^o, U^o , W^o for its biases, input weights and recurrent weights, respectively.

The process of training an LSTM involves performing the feedforward process to compute the activations and update the hidden state, and then performing the backpropagation process to adjust the weights and biases based on the error. This is done iteratively, with the model being trained on a large dataset and the weights and biases being updated after each pass through the data. The training process continues until the error between the predicted output and the true output is minimized to an acceptable level.

2.5 Overview of sentiment analysis and transformers: RoBERTa

Sentiment analysis is the process of analysing text data to determine the sentiment, or attitude, expressed in it. It is a subfield of natural language processing (NLP) that has gained significant attention in recent years due to the proliferation of online reviews, social media posts, and other forms of user-generated content. Transformers are a type of neural network architecture that was introduced in the paper "Attention is All You Need" (Vaswani, et al., 2017). They are particularly well-suited for processing sequential data, such as natural language text, and have achieved state-of-the-art results on a wide range of NLP tasks. In a transformer, the input data is processed by a series of self-attention layers, which allow the model to focus on different parts of the input at different times. The self-attention layers are followed by a series of fully connected layers, which process the output of the self-attention layers and produce the final output of the model. One key advantage of transformers is that they can process input sequences in parallel, rather than in a sequential fashion like traditional recurrent neural networks. This makes them much faster to train and allows them to handle longer input sequences more efficiently.

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing (NLP) model developed by Google. It was introduced in a paper titled "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," which was published in the journal arXiv in October 2018 (Devlin, Chang, Lee, & Toutanova, 2018). It is a type of transformer network that is trained using a technique called "masked language modelling," which involves masking a portion of the input text and predicting the masked tokens based on the context provided by the unmasked tokens. This allows BERT to learn contextual dependencies in the text and understand the relationships between words in a sentence. One of the key features of BERT is that it is "bidirectional," meaning that it considers the context on both the left and right sides of each token when processing the text. This contrasts with traditional NLP models, which typically process text in a left-to-right or right-to-left direction and are unable to capture contextual dependencies that span across multiple tokens. In this study a pretrained version of BERT called RoBERTa taken on Hugging Face has been used.

Hugging Face is a company that provides a platform for training and deploying natural language processing models. Hugging Face provides tools for using BERT and other NLP models in a variety of programming languages, including Python, JavaScript, and Swift.

RoBERTa (short for "Robustly Optimized BERT Approach") is a transformer-based language model developed by Facebook AI (Liu, et al., 2019). It has been pre-trained on a large dataset of English text and can be fine-tuned for a wide range of NLP tasks, such as language translation, summarization, and question answering.

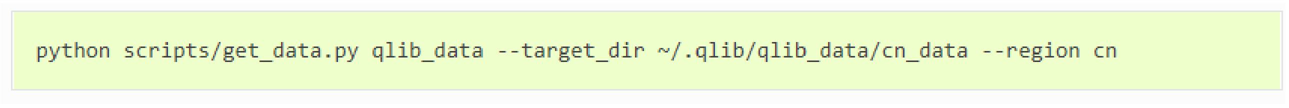
The most important difference between BERT and RoBERTa is that RoBERTa has been trained on a much larger dataset than BERT. This makes it more accurate and capable of performing better on some NLP tasks, but it also requires more computational resources to use. RoBERTa has also been trained using a different optimization method than BERT, which helps to reduce the amount of overfitting that can occur during fine-tuning. RoBERTa-base is a powerful and versatile NLP model that can be used for a wide range of tasks and is particularly well-suited for tasks that require a high level of accuracy.

The RoBERTa version chosen is a base model trained on 123.86 million tweets collected from January 2018 to December 2021 and finetuned for sentiment analysis with the TweetEval benchmark.

Methodology

3.1 Qlib data

In order to retrieve Qlib data, the “get_data.py” script has been used (Yang, Liu, Zhou, Bian, & Liu, 2020). The “get_data.py” script is a utility script that is used to retrieve financial market data from a remote server and store it in a local directory. The script is a command-line tool that is provided by the Qlib library to help users download and prepare financial data for use in their models. The script can be used to download data for a specific region and time interval, and it can also be used to update existing data. The version of the dataset can be specified when using the script to retrieve the data and the default version of the dataset is "v2", but it can be changed by passing the desired version as an argument to the script. Each version of the dataset has different features and data, and for the current dissertation’s project, the default version has been selected. The script has several parameters that can be changed to customize the data retrieval process. As shown in Figure 6, the main parameters include the target directory and the region: the target directory is the local directory where the data will be stored, while the region parameter is used to specify the financial market data, between American and Chinese market, that will be downloaded. In this dissertation we have chosen to forecast the CSI 300 market trend. The CSI 300 index is an indicator of the performance of Chinese stock market, and it includes the top 300 stocks traded on the Shanghai Stock Exchange and the Shenzhen Stock Exchange. The script uses the requests library to download data from the remote server, precisely from the default remote_URL provided at the start of the GetData class. The data retrieved by this URL are collected from Yahoo Finance, and we promptly selected the second version because it has a better quality than the other version. Moreover, data zip file is saved with the default name specified in QLIB_DATA_NAME variable.



```
python scripts/get_data.py qlib_data --target_dir ~/.qlib/qlib_data/cn_data --region cn
```

Figure 6: example of command to download stock data

Reference: Qlib (Microsoft, Github, 2020)

The dataset is downloaded in the form of a zip file, which is then unzipped and stored in the target directory. The script also has a built-in progress bar that displays the status of the download process, and it gives the chance to delete the existing Qlib data in the specified target directory. This last command is useful if we want to update the data contained in the target directory folder.

The data structure of Qlib data is divided into three main directories which are features, instruments, and calendars:

- The "features" directory contains the financial data of the instruments, such as open, close, high, low prices, volume, and trading factor, as shown in Figure 7.

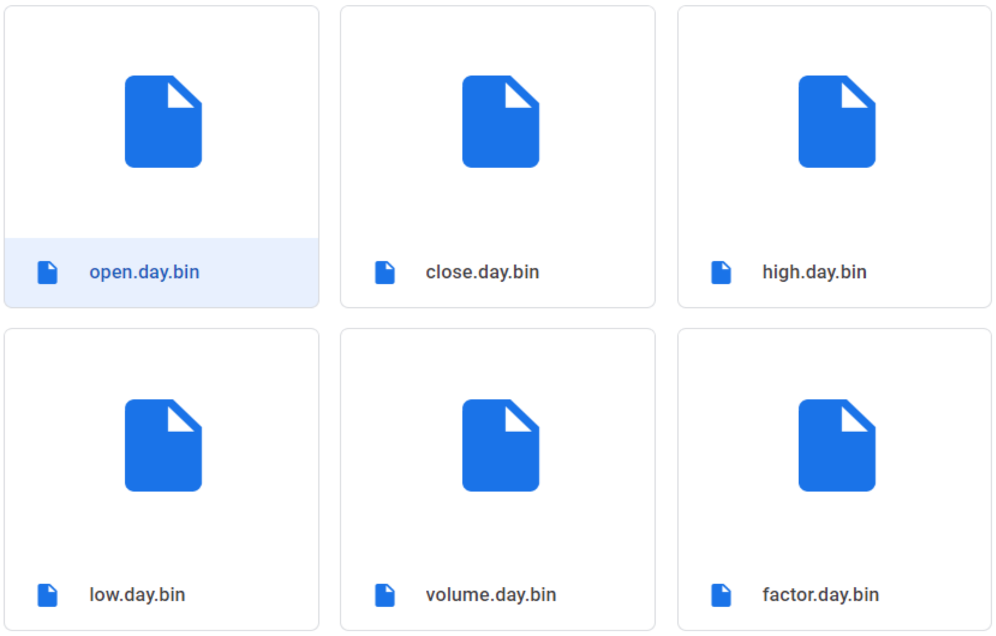
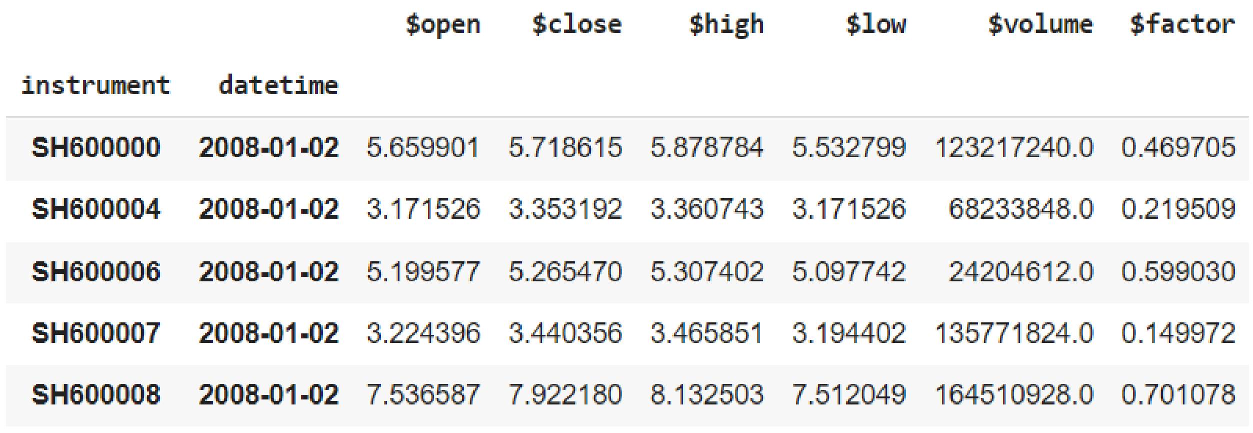


Figure 7: example of features directory of a stock

- The “instruments” directory contains the metadata of the instruments, such as their ticker symbol, industry, and exchange.
- The “calendars” directory contains the trading calendars, such as the dates of market open and close.

The dataset is stored in the binary format, and it is designed to be loaded into memory in a fast and efficient way.

Qlib data can also be uploaded in a pandas DataFrame (The pandas development team, 2020). It has a multi-level index, which makes it easy to work with and manipulate. As shown in Figure 8, the first level of the index contains the instrument code (stock market code), and the second level of the index contains the date.



		$open	$close	$high	$low	$volume	$factor
instrument	datetime						
SH600000	**2008-01-02**	5.659901	5.718615	5.878784	5.532799	123217240.0	0.469705
SH600004	**2008-01-02**	3.171526	3.353192	3.360743	3.171526	68233848.0	0.219509
SH600006	**2008-01-02**	5.199577	5.265470	5.307402	5.097742	24204612.0	0.599030
SH600007	**2008-01-02**	3.224396	3.440356	3.465851	3.194402	135771824.0	0.149972
SH600008	**2008-01-02**	7.536587	7.922180	8.132503	7.512049	164510928.0	0.701078

Figure 8: Dataset structure

The columns fields represent the financial indicators used for the portfolio forecasting (Microsoft, Component data, 2020). In particular:

- Open: the adjusted opening price at which a stock or other financial instrument first trade on an exchange when the market opens
- Close: the adjusted closing price at which a stock or other financial instrument last trade on an exchange when the market closes.
- High: the highest adjusted traded price for a stock or other financial instrument during a specific timeframe (in this case a whole day).
- Low: the lowest adjusted traded price for a stock or other financial instrument during a specific period.
- Volume: the number of shares or other units of a financial instrument that were traded during a specific timeframe.
- Factor: Factor prices are used to adjust the market price of a financial instrument for various types of corporate actions, such as stock splits or dividends. They are called also Restoration factor prices (Hull, 2018).

In summary, the dataset downloaded using the “get_data.py” script resumes the stock market indicators of CSI 300 market from the 1st of August 2008 to 25th of

September 2020. We decided to include the first part of the covid period in order to understand if the LSTM could predict an event catastrophic such as the pandemic. It contains 682 stocks, due to the fact that from 2008 to 2020 the stocks belonging to the CSI 300 market have been systematically changed. After a phase of checking the dataset, as the presence of missing values or errors, we found out that the quality data was good enough to not need an extra special cleaning. The high quality of the dataset is justified because the Qlib repository provides a wide range of data pre-processing tools and functions, including functions for cleaning and transforming data, as well as functions for feature scaling, normalization, and dimensionality reduction.

3.2 Twitter data

Snscrape (JustAnotherArchivist, 2018) is a python package that allows users to scrape various social media platforms, such as Twitter, Instagram, and Reddit. It is built on top of the popular scraping library, BeautifulSoup (Richardson, 2007), and provides an easy-to-use interface for users to extract data from different websites. Beautiful Soup is a Python package for parsing HTML and XML documents, creating parse trees from these documents for easy traversal and manipulation. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree, and it sits on top of popular Python parsers like lxml and html5lib, allowing users to try out different parsing strategies or trade speed for flexibility. For these reasons it is often used for web scraping, data mining, and data analysis.

One of the main features of snscrape is its ability to scrape tweets from Twitter. The package provides a module specifically for this purpose called "twitter". This module allows users to retrieve tweets from a specific account or a hashtag, as well as filter the tweets by certain criteria such as the number of retweets or the presence of certain words in the tweet, and this allows users to easily extract a specific subset of tweets that are relevant to their research. Additionally, snscrape allows users to save the scraped data in various formats, such as JSON, CSV, or even SQLite databases. This makes it easy for users to load and analyze the data using other tools or libraries.

In this study, we have used snscrape package to scrape tweets from 10 specifics twitter accounts from the 1st of August 2008 to today. We have chosen to scrape tweets from 5 twitter accounts that usually tweet financial news, and 5 other accounts that usually report general news. The reason behind this choice is that we wanted to have a dataset that contains a diverse range of news and information. Financial news play an important role in this research, as they relate to the stock market and other financial markets. On the other hand, general news can also be considered important because they can provide a context for the financial news previously mentioned. For example, a political event may have an impact on the stock market, and by having both types of news in the dataset we could be able to study and analyse the correlation between them two.

For each account we scraped the Username, the date and the content of the tweets, and in order to make the cleaning and sentiment analysis more manageable, we retrieve the tweets in 10 separate DataFrames. This way, data pre-processing was easier and more efficient, and sentiment analysis with BERT was faster.

3.3 Pre-processing of Twitter data

Pre-processing tweet data is a crucial step in natural language processing, as it allows for the efficient and effective analysis of large amounts of data. Before analysing the sentiment of tweets, it is necessary to clean and prepare the data for the model to be able to understand it. The first step in pre-processing tweet data is to remove any irrelevant information such as URLs, mentions, and hashtags. We removed mentions and URLs to protect the privacy of users and sensitive data, by replacing mentions with a generic tag. This is also done to reduce the noise in the data and make it easier for the model to understand the context of the text. Moreover, we checked for any duplicate tweets, but fortunately the snscrape package didn’t return this type of errors. After that, the tweets have been tokenized and normalized: tokenization is the process of breaking down the text into individual words or phrases, while normalization includes converting all text to lowercase, removing any punctuation, and removing any stop words. The pre-processed tweets have been tokenized using the AutoTokenizer from the Transformers library (Wolf, et al., 2020), which is specifically designed to work with RoBERTa models.

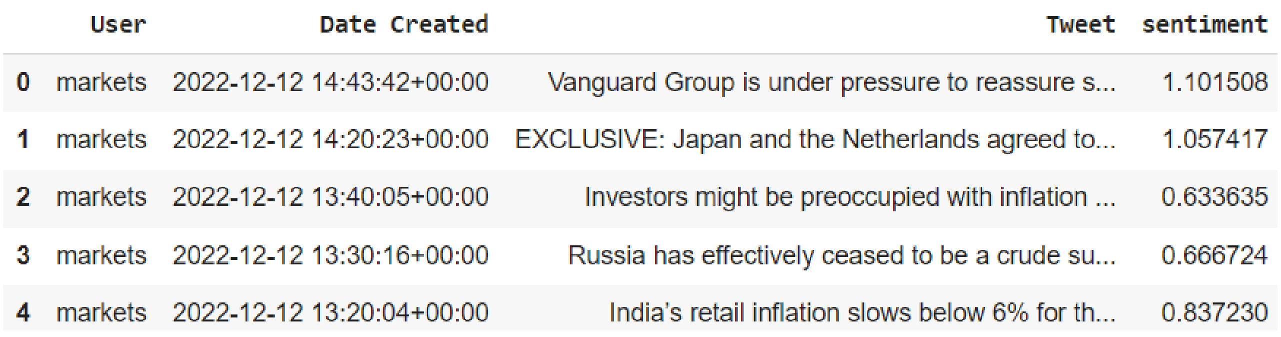
3.4 RoBERTa base for Sentiment Analysis

RoBERTa-base is a pre-trained transformer model developed by Facebook AI Research (FAIR) that was trained on a massive amount of data, specifically on a diverse set of internet text in order to improve the performance of NLP tasks (Liu, et al., 2019). The model is a variant of the popular BERT model, but with some modifications in the training process that lead to improved performance on a wide range of natural language understanding tasks.

The Twitter-roBERTa-base model is specifically pre-trained on twitter data and fine-tuned on sentiment analysis tasks, which makes it well suited for tasks such as sentiment classification, emotion detection and sarcasm detection on twitter data. The RoBERTa-base model uses a deep neural network with a transformer architecture, which allows it to handle input sequences of varying lengths and can be fine-tuned on a wide range of NLP tasks with relatively little task-specific training data. In this study, we used the "cardiffnlp/twitter-roberta-base-sentiment-latest" (Loureiro, Barbieri, Neves, Anke, & Camacho-Collados, 2022) available on Hugging Face (Face, 2016), that is a version of the RoBERTa model for sentiment analysis on tweets. This version of RoBERTa has been pre-trained on a large dataset of tweets, and it is specifically fine-tuned for sentiment analysis, which makes it optimal for this task.

The pre-training process of RoBERTa operates on a larger batch size and learning rate, and a longer training period than BERT, which allows it to learn more robust representations of the input data. Additionally, it deploys a dynamic masking, which improves the model's ability to learn from the input data by masking a random 15% of the tokens in the input sequence, rather than the 10% used in the BERT model (Bahdanau, Cho, & Bengio, 2014). This pre-training allows the model to understand the context and nuances of language frequently used in tweets, making it a suitable choice for sentiment analysis. It is important to note that pre-processing tweet data is a time-consuming and iterative process, as the specific needs of each dataset and analysis may vary. For this reason, the RoBERTa-base model has been fine-tuned on a smaller dataset of tweets, which allows it to learn the specific features of the data and improve its performance. The fine-tuning process is done by adjusting the model's parameters to better fit the task of sentiment analysis.

We used the tokenizer tweets as input for the model, receiving a prediction for the sentiment of the text. As you can see in Figure 9, we added a new column named ‘sentiment’ that highlight the score of each Tweet.



	User	Date Created	Tweet	sentiment
0	markets	2022-12-12 14:43:42+00:00	Vanguard Group is under pressure to reassure s...	1.101508
1	markets	2022-12-12 14:20:23+00:00	EXCLUSIVE: Japan and the Netherlands agreed to...	1.057417
2	markets	2022-12-12 13:40:05+00:00	Investors might be preoccupied with inflation ...	0.633635
3	markets	2022-12-12 13:30:16+00:00	Russia has effectively ceased to be a crude su...	0.666724
4	markets	2022-12-12 13:20:04+00:00	India’s retail inflation slows below 6% for th...	0.837230

Figure 9: example of tweet DataFrame after sentiment analysis with RoBERTa model

The model outputs the scores in a range between 2 and 0 for each of the possible sentiments (positive, neutral, negative), where the maximum positive sentiment is equal to 2, the neutral sentiment to 1, while the negative sentiment equal to 0. In this way it is possible to calculate the daily average sentiment for each account and then the daily average sentiment gained by the union of the ten accounts.

3.5 Enrichment of Qlib data

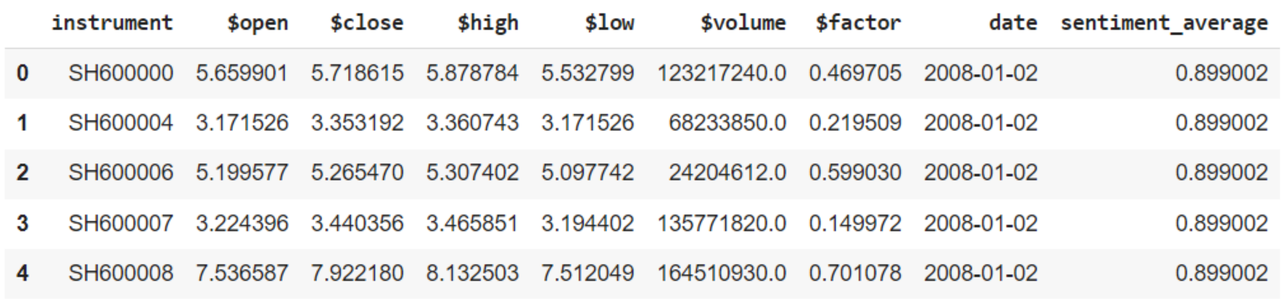
Enrichment refers to the process of adding new information, context, or value to existing data. In the context of quantum computing, data enrichment can involve a variety of techniques, such as:

- Feature engineering: adding new features or attributes to the data, based on existing features or external information

- Data augmentation: generating new data samples by applying transformations or perturbations to existing data samples.
- Annotation: adding semantic labels or other annotations to the data, to provide additional context or information.
- Domain adaptation: adapting data from one domain or application to another, by adding new features or adjusting existing features.
- Data fusion: combining multiple data sources or modalities to create a more complete or representative dataset.

Enriching data can help to improve the performance of quantum machine learning and optimization algorithms, by providing more information or context about the problem or the data.

In this dissertation, the Qlib financial dataset has been enriched with the daily sentiment average calculated on daily tweets. In order to concatenate the new feature, we adapted the datetime format of the date column in the tweet dataset equal to the format available in the Qlib data. As you can see in Figure 10, the final Qlib dataset presents the 6 features previously showed and the sentiment feature.



	instrument	$open	$close	$high	$low	$volume	$factor	date	sentiment_average
0	SH600000	5.659901	5.718615	5.878784	5.532799	123217240.0	0.469705	2008-01-02	0.899002
1	SH600004	3.171526	3.353192	3.360743	3.171526	68233850.0	0.219509	2008-01-02	0.899002
2	SH600006	5.199577	5.265470	5.307402	5.097742	24204612.0	0.599030	2008-01-02	0.899002
3	SH600007	3.224396	3.440356	3.465851	3.194402	135771820.0	0.149972	2008-01-02	0.899002
4	SH600008	7.536587	7.922180	8.132503	7.512049	164510930.0	0.701078	2008-01-02	0.899002

Figure 10: head of Qlib dataset enriched with daily sentiment

Thanks to the Qlib repository, the dataset columns have been converted into binary files and saved in the subdirectories of each stock. In this way, it can be loaded easily into memory and it can be used by the LSTM neural network in a fast and efficient way.

3.6 LSTM structure

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) that is designed to handle the problem of vanishing gradients in traditional RNNs. The problem of vanishing gradient in RNNs occurs when the gradients of the error function with respect to the model's parameters become very small during the backpropagation process, making it difficult for the optimizer to update the parameters and improve the model's performance (Goodfellow, Bengio, & Courville, 2016). This happens because the gradients are multiplied by the weight matrix of the recurrent connections many times during the backpropagation, causing them to decrease exponentially with the number of layers. This problem can be mitigated by using techniques such as gradient clipping, or by using architectures such as LSTMs or GRUs which have a gating mechanism that can help to preserve the gradients during backpropagation. Using a gating mechanism, LSTMs have an additional memory cell, input, forget and output gates, and a hidden state that allow them to selectively remember or forget information from previous time steps.

For the purpose of implementing the LSTMs, we started by defining the LSTM class which takes several hyperparameters as input:

- Input dimension: called d_feat, it refers to the number of features in the input data that is being fed into the network at each time step.

- Hidden size: the hidden size refers to the number of units in the LSTM cell's hidden state. It is a hyperparameter that determines the capacity of the network to retain information from previous time steps, and it’s typically set to a small number of neurons, often ranging from 64 to 256, depending on the complexity of the problem.
- Number of layers: the LSTM architecture is designed to allow information to flow through the network for a certain number of time steps, known as the number of layers.
- Dropout rate: it’s the probability that a neuron in the LSTM network will be "dropped out" or ignored during training. This is a regularization technique used to prevent overfitting by randomly dropping out neurons in the network during training. The dropout rate is a value between 0 and 1, where a value of 0 means no neurons will be dropped out and a value of 1 means all neurons will be dropped out.
- Number of trainings epochs: relates to the number of times the entire training dataset is passed through the LSTM network during the training process. One epoch is completed when the LSTM has seen all the samples in the training dataset once. The number of training epochs is a hyperparameter that can be adjusted to control the training duration of the LSTM model. A higher number of epochs typically leads to a more trained model, but also increases the risk of overfitting.
- Learning rate: it’s a hyperparameter that determines the step size at which the optimizer makes updates to the model's parameters during training. It is a scalar value that is used to scale the update step of the optimizer. A smaller learning rate means that the model's parameters will be updated less frequently, while a larger learning rate means that the parameters will be updated more frequently.

- Evaluation metric: an evaluation metric is a way to measure the performance of the model during the training process. This metric is used to evaluate how well the model is performing on a given dataset and to monitor the progress of the training. Common evaluation metrics used in LSTM include accuracy, precision, recall, F1 score, area under the ROC curve (AUC) or Mean Squared Error (MSE). Aiming to evaluate the performance of the models we set the Mean Squared Error loss function, that is a commonly used loss function for regression problems. It measures the average of the squared differences between the predicted values and the true values. MSE is calculated by taking the average of the squared difference between the predicted values and the actual values.
- Batch size: the batch size related to the number of training examples used in one forward/backward pass. The batch size is a hyperparameter that can be adjusted to trade-off between memory usage and training speed. Using a larger batch size can speed up training, but it also requires more memory. Instead, a smaller batch size can use less memory, but it requires more iterations to achieve the same level of training as a larger batch size.
- Optimizer: it's an algorithm that is used to adjust the parameters of the model to minimize the loss function. The default Qlib optimizers include the gradient descent algorithm and the Adam (Adaptive Moment Estimation) algorithm. I chose to use the Adam optimizer, that is a popular optimization algorithm for training neural networks. It exploits a combination of the gradient of the loss function with respect to the parameters and the historical gradient information to adapt the learning rate for each parameter. This optimizer work by iteratively updating the weights of the LSTM model intending to reduce the difference between the predicted output and the true output.

- GPU: represents the GPU id used for training the model.

After that, I defined the LSTMModel class, which inherits from the PyTorch nn.Module class (Paszke, et al., 2019). This class contains the architecture of the LSTM model, including the input layer, LSTM layers, and output layer. The input layer takes in the dimensional input at each time step, and the LSTM layers process the data with the specified hidden size and number of layers. To train the LSTM model, we used the fit method. This method takes in the training data and trains the model for the specified number of epochs, along this study we set this number equal to 200. We also included early stopping, which monitors the performance of the model on the validation data and stops training if the performance on the validation data has not improved for 20 consecutive epochs. Finally, the output layer produces the final prediction for the input sequence.

3.7 Qlib workflow

The Qlib repository has a command line tool called "qlib_run" which can take a workflow file as an input and run the pipeline specified in the file. The "qlib_run" command will execute the pipeline in the order specified in the file, running each task one by one. The pipeline can include tasks such as data processing, model training, backtesting, and performance analysis.

The workflow is a configuration file developed using the Qlib library. The workflow resumes all the parameters set up with the aim to predict the optimal portfolio, and it includes the following components:

- qlib_init: initializes the Qlib library and sets the data provider folder path and the cn region.

- market and benchmark: I’ve set the csi 300 market and the CSI300 index benchmark to be used in the strategy.
- data_handler_config: this component defines the parameters for the data handler, which is responsible for loading and processing the data. It specifies the start and end dates for the data, and the instruments to be used, and the creation of the label. The label is the output or target variable that the network is trying to predict. In my dissertation the label has been calculated as follow:

  ["Ref($close, -2) / Ref($close, -1) - 1"],

  this formula is used to calculate the return of an asset over a two-day period. This return value will be used as the label for training the LSTM networks.
- port_analysis_config: it resumes the parameters for the portfolio analysis, which includes the TopkDropout strategy settled with 50 stocks in portfolio and a specific number of daily drop equal to 5. After that, it contains the backtest parameters, such as start and end time, amount of virtual money available and the account information.
- task: this part contains the specific tasks to be executed in the workflow. First of all, it recalls the script of the LSTM model with the kwargs arguments. The difference between the models’ parameters is given by the dimension feature parameter. For the benchmark model we set 6 features, while in order to implement the LSTM model with the sentiment score, we changed the number of input dimension equal to 7. Moreover, the task recalls the script of the dataset handler. Here we have used the Qlib Alpha360 handler for the benchmark model, while we deploy a

different version aiming to prepare and import the enriched Qlib data. Alpha360 is responsible for loading the data and applying the preprocessing steps defined in the data_handler_config. Finally, the recorders have been set, where the SignalRecord, SigAnaRecord and PortAnaRecord classes are configurated to show and save the results.

Results

4.1 Performance portfolio metrics

To the extent of evaluating the models' performances, we have deployed different kinds of analysis. We have utilized the Intraday Trading component of the Qlib's library, that uses the prediction scores associated to each stock to return the backtest results. The backtest results are divided in excess return without and with cost of trading, and they are composed by:

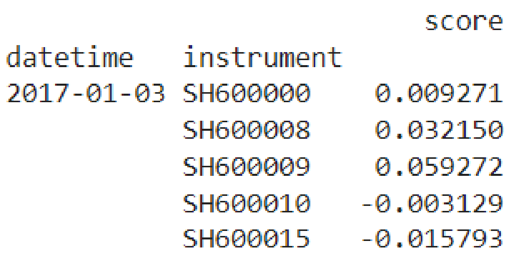
- Mean value of the cumulative abnormal return without (or with) cost.
- Standard deviation of cumulative abnormal return without (or with) cost.
- Annualized rate of cumulative abnormal return without (or with) cost.
- Information ratio without (or with) cost.
- Maximum drawdown of cumulative abnormal return without (or with) cost.

Given that a numerical analysis has limited capability to explain the results, the Intraday Trading component also helps users to evaluate and analyze investment portfolios visually. The graphics employed to explore the results are split into analysis position graphs and analysis model graph. The analysis position component includes report, information ratio and cumulative return graphs, while the analysis model component incorporates the model performance graph. All of the accumulated profit metrics (e.g. return, max drawdown) in Qlib are calculated by summation. This avoids the metrics or the plots being skewed exponentially over time.

4.2 LSTM results

The model has been trained using the prepared financial dataset without the sentiment feature. The dataset has been split in training, validation, and test set: the training set is from 2008-01-01 to 2014-12-31, the validation set is from 2015-01-01 to 2016-12-31 and the test set is from 2017-01-01 to 2020-08-01. This approach allowed us to train the model on the training set, bringing into play the validation set to optimize the model, and finally testing it on the unseen test set.

Remembering that the early stopping was set to 20 epochs, the model has been trained and validated for 52 epochs and then it reached the early stopping criteria. The best Mean Squared Error obtained by the model was equal to 0,992580 reached at the 32nd epoch. After the training and validation process, the prediction scores for each stock have been generated as a part of the evaluation of the model's performance, as shown in Figure 11.

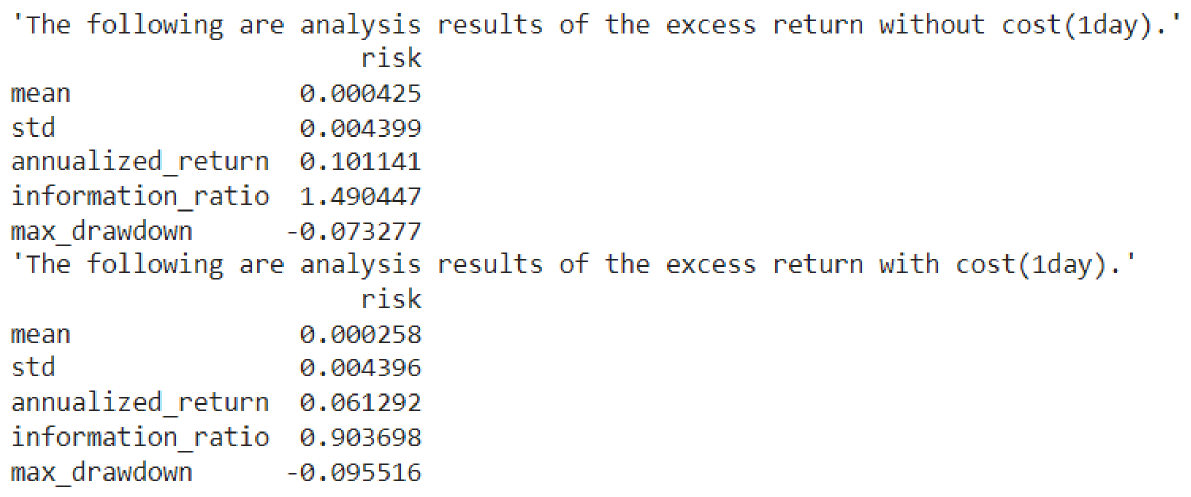


```
                           score
datetime   instrument
2017-01-03 SH600000     0.009271
           SH600008     0.032150
           SH600009     0.059272
           SH600010    -0.003129
           SH600015    -0.015793
```

Figure 11: example of prediction scores after training process

These prediction scores have been used as input for the test process, that is the final step in the evaluation of the model. The test process exploits the prediction scores to generate trading signals and assess the performance of the model under live market conditions, using the testing dataset.

The numerical backtest results are shown in Figure 12:



```
'The following are analysis results of the excess return without cost(1day).'
                       risk
mean                0.000425
std                 0.004399
annualized_return   0.101141
information_ratio   1.490447
max_drawdown       -0.073277
'The following are analysis results of the excess return with cost(1day).'
                       risk
mean                0.000258
std                 0.004396
annualized_return   0.061292
information_ratio   0.903698
max_drawdown       -0.095516
```

Figure 12: backtest results of LSTM model

These results are the performance metrics of the TopKdropout portfolio strategy. The metrics are calculated based on the excess return of the portfolio, which is the return of the portfolio minus the return of the benchmark. The first set of results shows the performance of the portfolio strategy without considering trading costs. The mean return is 0.000425, and the standard deviation of returns is 0.004399, indicating a relatively high level of volatility. The annualized return is 0.101141, which is the average return of the strategy over a year. The information ratio is 1.490447, which is calculated comparing the excess return of the portfolio to the benchmark's excess return divided by the tracking error. The max drawdown is -0.073277, which is the maximum drop period of the portfolio's value.

The second set of results shows the performance of the portfolio strategy taking trading costs into account. The mean return is 0.000258, and the standard deviation of returns is 0.004396. The annualized return is 0.061292, while the information ratio is 0.903698. The max drawdown in this case is -0.095516. In pursuance of understanding how the investment strategy works, one example of daily positions has been provided in Figure 13.

```
Timestamp('2018-01-04   00:00:00'):   {'_settle_type':   'None',
'position':  {'cash':  598059.5864510699,  'now_account_value':
139564710.01267228,  'SH600666':  {'amount':  305335.3983153503,
'price':   6.992496490478516,   'weight':   0.015297969672599087,
'count_day':  189},  'SH600015':  {'amount':  244467.59698539908,
'price':   7.2934417724609375,   'weight':  0.012775508820994652,
'count_day':   111},   'SH601166':  {'amount':  634411.7679126329,
'price':   3.2975053787231445,   'weight':  0.014989292184444175,
'count_day': 110},etc.
```

Figure 13: example of portfolio timestamp

As we can see, the daily positions report different key information: cash represents the part of initial cash still available. The now account value represents the portfolio’s value taking cash available and cash invested into account. After that, all the shares held are reported, resuming the amount of stocks held, the daily price, the initial cash portfolio’s weight and the count day, that tracks for how long a specific stock belonged to our portfolio.

After a numerical analysis, we studied more deeply the backtest results using a graphical approach. In Figure 14 the backtest report has been displayed:

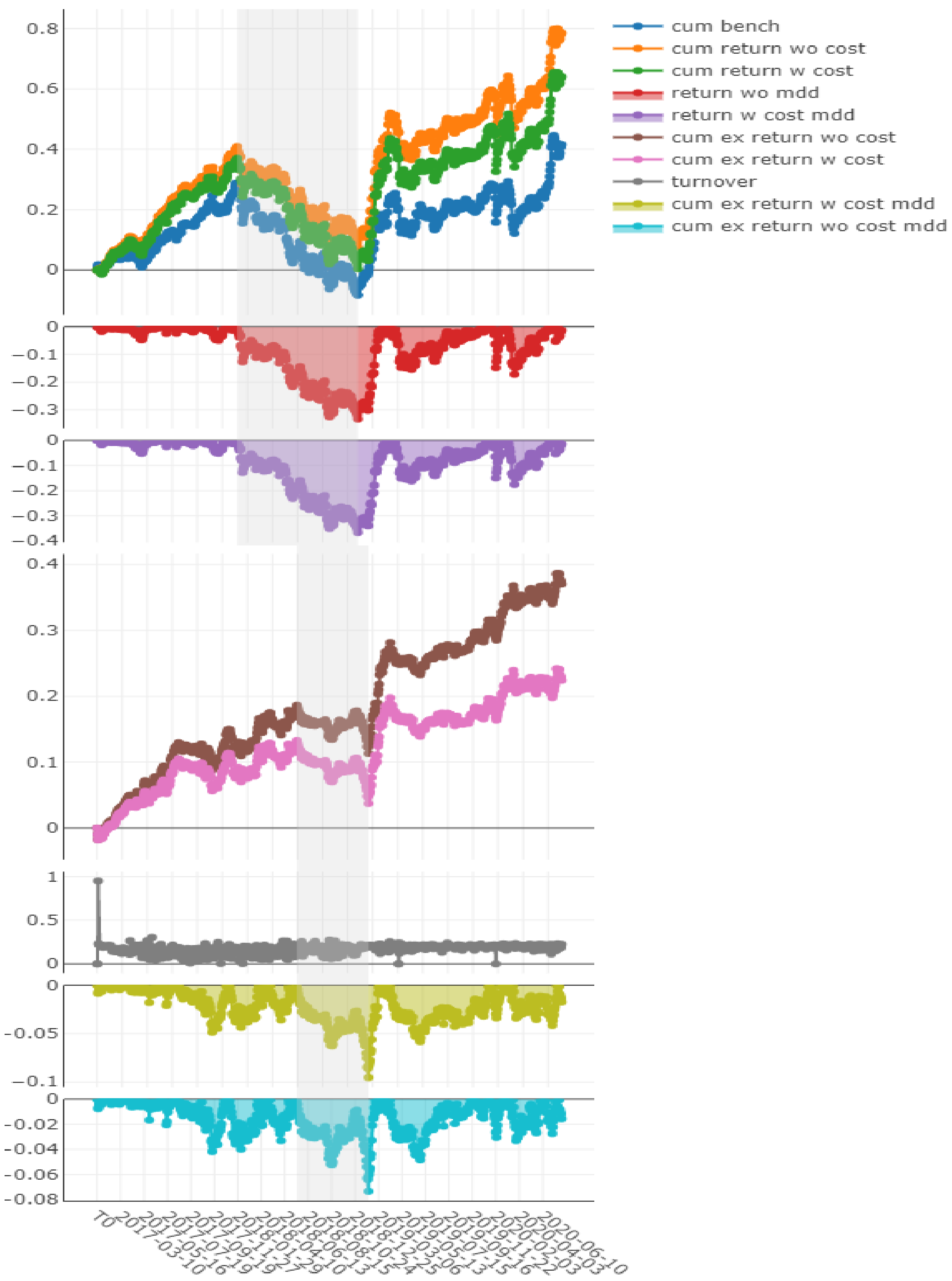


Figure 14: analysis position report graph LSTM model

The parameters considered to assess the portfolio performance include different time series:

- cum bench: cumulative returns series of benchmark
- cum return wo cost: cumulative returns series of portfolio without cost
- cum return w cost: cumulative returns series of portfolio with cost
- return wo mdd: maximum drawdown series of cumulative return without cost
- return w cost mdd: maximum drawdown series of cumulative return with cost
- cum ex return wo cost: the CAR (cumulative abnormal return) series of the portfolio compared to the benchmark without cost.
- cum ex return w cost: the CAR (cumulative abnormal return) series of the portfolio compared to the benchmark with cost.
- Turnover: turnover rate series
- cum ex return wo cost mdd: drawdown series of CAR (cumulative abnormal return) without cost
- cum ex return w cost mdd: drawdown series of CAR (cumulative abnormal return) with cost

The cumulative that returns series of a benchmark, is a measure of the performance of a particular investment benchmark, such as an index or a market, over a certain interval. It represents the total return of the benchmark, including both capital appreciation and dividends, as a percentage of the initial investment. The difference between the cumulative shows a series of benchmark and the cumulative returns series of a portfolio with or without cost, this is a measure of the performance of the portfolio in relation to the benchmark. If the portfolio has a higher cumulative return than the benchmark, it has outperformed the benchmark, whereas if the portfolio has a lower cumulative return than the benchmark, it has underperformed the benchmark. As we can see, the LSTM model outperformed the benchmark.

Maximum drawdown series of cumulative return with or without cost refers to the calculation of the largest peak-to-trough decline in the value of a portfolio or investment over a specific period. This calculation is typically used to measure the risk of a portfolio or investment, as it represents the largest loss that an investor would have experienced if they had invested at the highest point and then sold at the lowest point of the portfolio's or investment's value. In case of portfolio without cost the higher loss faced is equal to -0,333, while for a portfolio with cost is equal to -0,367.

The CAR series of the portfolio compared to the benchmark is a measure of the performance of a portfolio relative to a benchmark, such as a market index. It is calculated by subtracting the cumulative returns of the benchmark from the cumulative returns of the portfolio, and then cumulatively summing the differences. This allows for an evaluation of how much better (or worse) the portfolio performed compared to the benchmark, considering any costs associated with the portfolio. The CAR is often used as a measure of the skill or alpha of a portfolio manager, as it isolates the returns that are due to the manager's decisions, as opposed to returns that are simply due to the overall market conditions. In this graph we can see better that in general the portfolio with or without cost has outperformed the benchmark index.

The turnover rate series is a metric that measures the rate at which the securities in a portfolio are bought and sold over a specific period of time. Analyzing the graph we can see a constant turnover rate, due to the TopKdropout strategy implemented.

The drawdown series of CAR with or without cost is a measure of the decline in the portfolio's CAR from its historical peak. It represents the largest percentage loss from a peak to a trough of the CAR, before a new peak is reached. Due to the trading cost, the maximum drawdown of portfolio with trading cost compared to benchmark, equal to -0,095, is higher than the portfolio without cost compared to benchmark, equal to -0,073.

In the interest of studying deeply the risk trend of the portfolios built by LSTM model against the benchmark trend, the information ratio has been displayed in Figure 15:

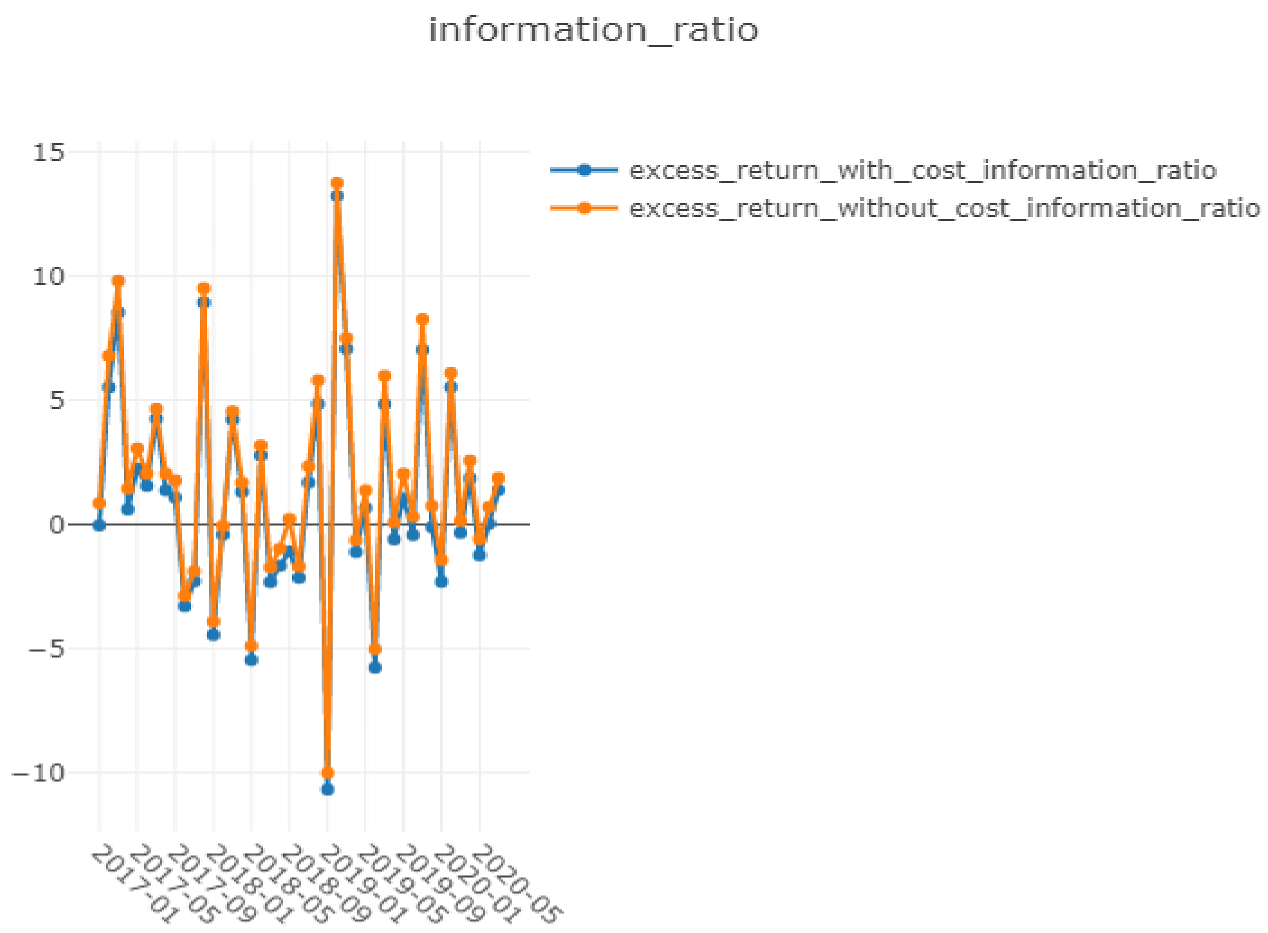


Figure 15: information ratio trend between LSTM models and benchmark

The Information Ratio provides the amount of portfolio outperformance relative to the benchmark for each unit of relative risk (represented by the tracking error) and allows an assessment of the manager's ability to outperform the benchmark relative to the risk taken (represented by the deviation from the benchmark). As we can see, the information ratio trend underlines the outperformed results of LSTM models against the benchmark, resulting in a positive trend most of the time.

With the purpose of studying the volatility of portfolio, the standard deviation series as been plotted in Figure 16:

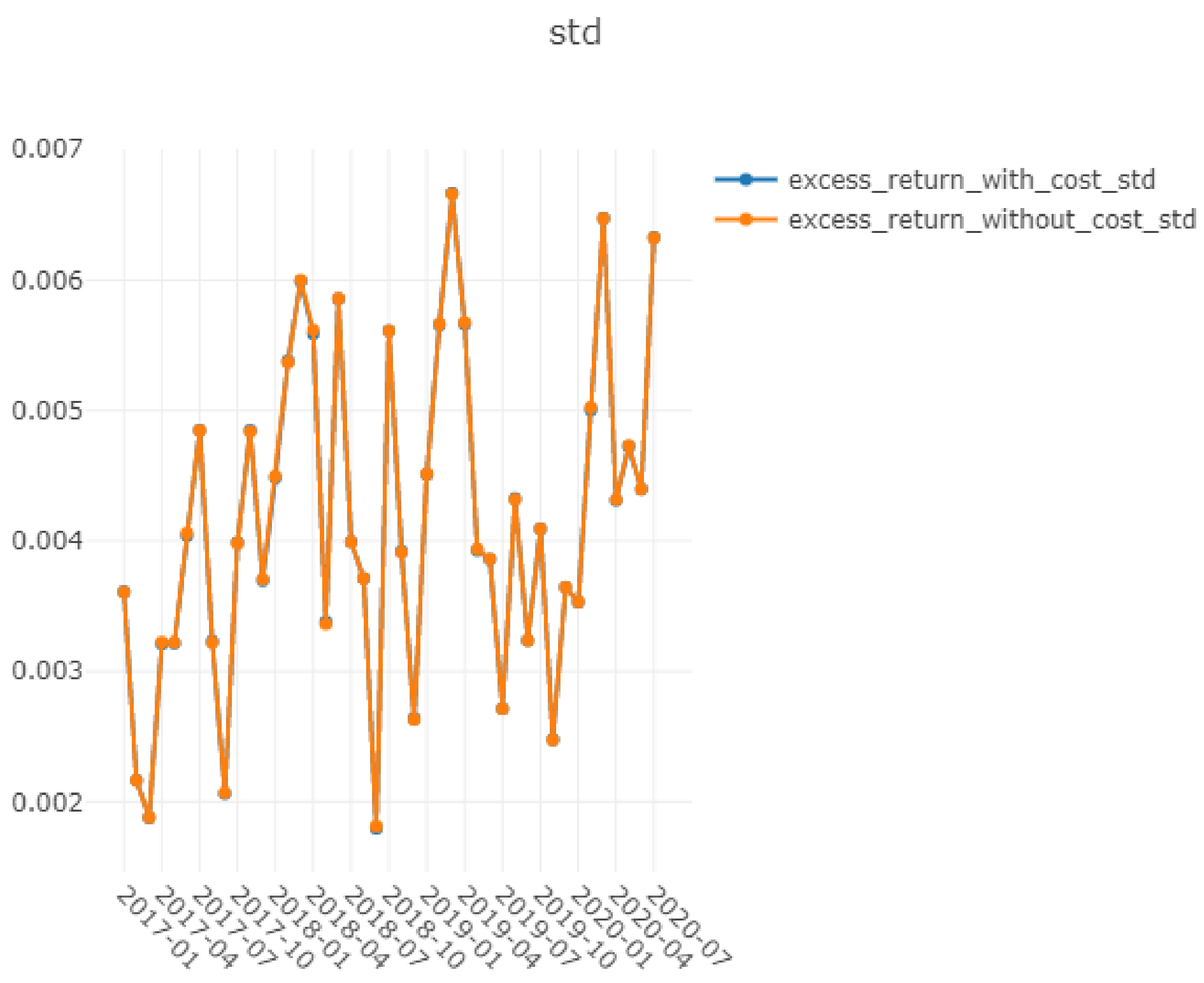


Figure 16: standard deviation time series LSTM model

This graph reports the standard deviation series of monthly cumulative abnormal return. As we can see the standard deviations aren’t constant and it means that the portfolios are affected by high volatility. Moreover, we can see that the excess returns with and without trading cost have a similar volatility: this is expected given that the trading strategy doesn’t take the trading cost into account.

Finally, a new indicator has been calculated: the ranking ratio. The ranking ratio is a metric used to evaluate the performance of a stock selection model, and it’s calculated as follow:

Ranking ratio = Ascending ranking of label/Number of stocks in the portfolio,

a lower ranking ratio indicates that the portfolio is performing better than the benchmark, as the ranking of the portfolio's label is lower than the number of stocks in the portfolio.

In Figure 17 we can see the cumulative return series for different groups of shares and the cumulative return series for long-short and long-average positions.

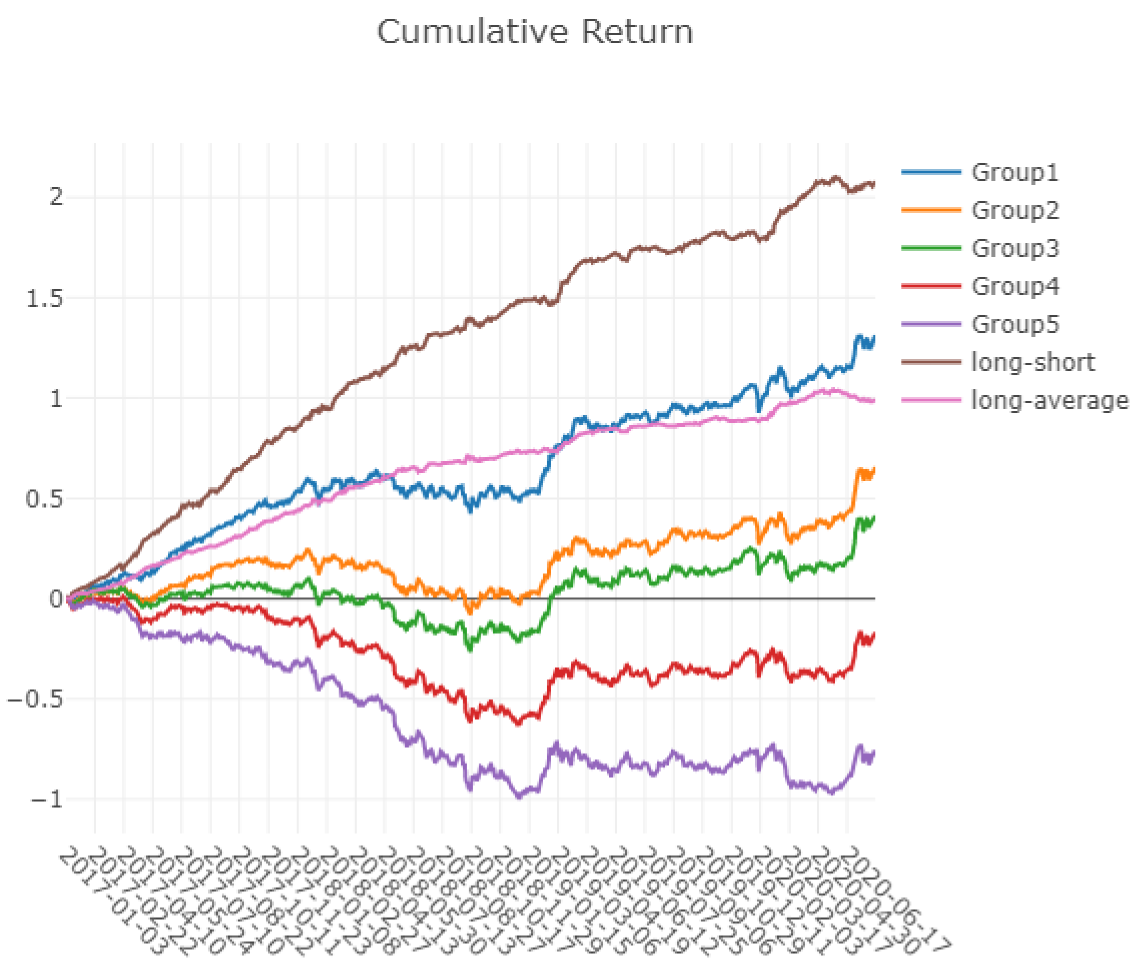


Figure 17: LSTM model performance on cumulative return series

The groups compositions are the following:

- Group1: cumulative return series of stocks with ranking ratio of label <= 20%.
- Group2: cumulative return series of stocks with ranking ratio of label between 20% and 40%.
- Group3: cumulative return series of stocks with ranking ratio of label between 40% and 60%.
- Group4: cumulative return series of stocks with ranking ratio of label between 60% and 80%.

- Group5: cumulative return series of stocks with ranking ratio of label > 80%.

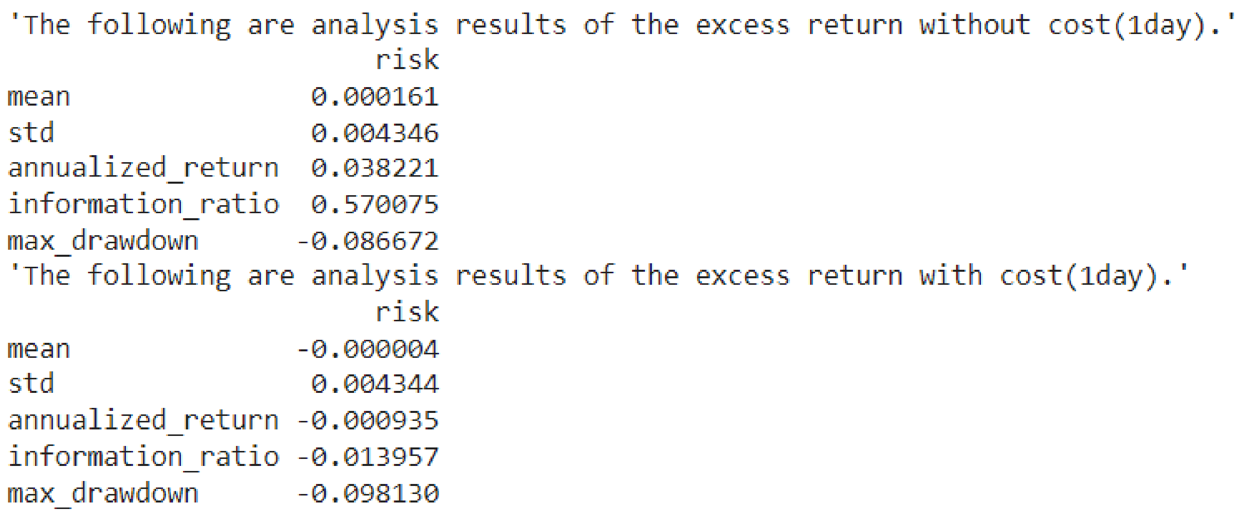
The long-short series is the difference between the cumulative return of Group1 and Group5, while the long-average series is the difference between the cumulative return of Group1 and the average cumulative return for all stocks.

As we can see the investment portfolio has always a positive trend in both long-short and long-average time series. Moreover, four of the groups had generally a positive trend during the test, showing again than the LSTM models outperformed the benchmark. Generally, these results showed that the strategy had a good performance, with a high information ratio and a low max drawdown, but it has a high volatility, and the returns are lower when considering trading costs.

4.3 Sentiment Qlib data results

Using the financial dataset enriched with the sentiment feature, the Qlib LSTM model has been trained, validated and tested. The same training, validation and testing division have been settled, and the model has been trained and validated for 57 epochs before reaching the early stopping criteria. The best Mean Squared Error gained by the model was equal to 0,993100 reached at the 37th epoch.

The backtest results has been resumed in Figure 18:



```
'The following are analysis results of the excess return without cost(1day).'
                       risk
mean               0.000161
std                0.004346
annualized_return  0.038221
information_ratio  0.570075
max_drawdown      -0.086672
'The following are analysis results of the excess return with cost(1day).'
                       risk
mean              -0.000004
std                0.004344
annualized_return -0.000935
information_ratio -0.013957
max_drawdown      -0.098130
```

Figure 18: backtest results of Qlib LSTM model

Analyzing the first set of results we can see that the main return is equal to 0,000161 and the standard deviation is equal to 0,004346. The annualized return is equal to 0,038221, while the information ratio is 0,570075. Finally the max drawdown is equal to -0,086672.

Instead, the second set of results reported a negative mean return, equal to -0,000004. As expected, the standard deviation is really similar to the first set standard deviation, equal to 0,004344. The annualized return resulted to be negative and equal to -0,000935. The information ratio is equal to -0,013957 and the max drawdown equal to -0,098130.

From this numerical analysis we can deduct that the Qlib LSTM model doesn’t show real improvements. In furtherance of confirming the numerical results, graphical reports have been displayed. The Qlib LSTM model backtest results have been shown in Figure 19:

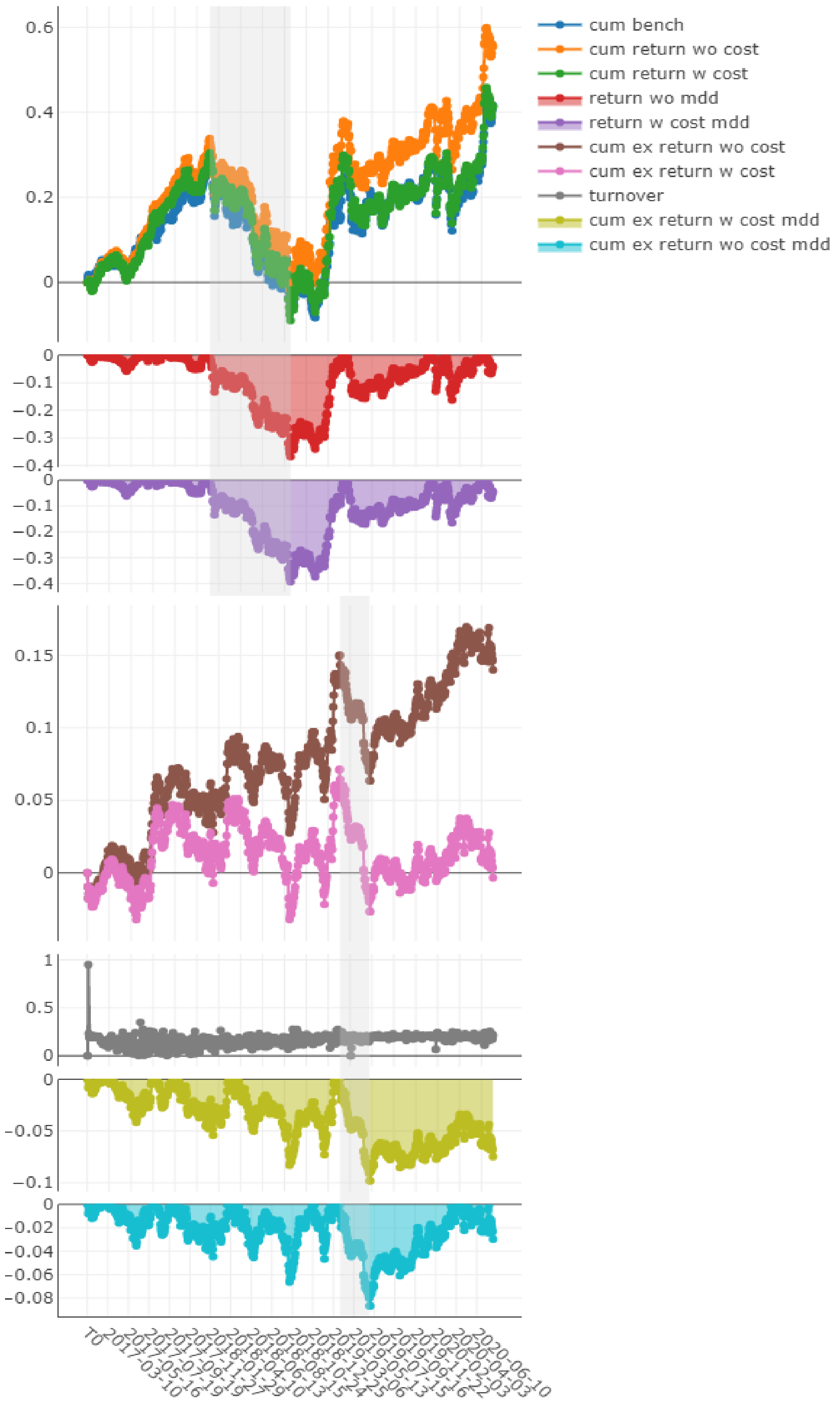


Figure 19: analysis position report graph Qlib LSTM model

Studying the cumulative return series, it’s not clear if the Qlib model outperformed the benchmark. More in detail, if we can assume that the cumulative return series of the portfolio without trading cost outperformed the benchmark series, it’s not clear if the cumulative return series with trading cost had outperformed the benchmark. Looking at the maximum drawdown series, we can notice that the highest loss reached by the portfolio without cost is equal to -0,3687, while for the portfolio with cost is equal to -0,3936.

The cumulative abnormal series of the portfolio resume what we already saw studying the cumulative return series: the excess return series of the portfolio without cost can relatively outperform the benchmark series, while the excess return series of portfolio with trading cost can’t outperform the benchmark series constantly. The maximum drawdown series of cumulative abnormal with cost is equal to -0,098, while without cost is equal to -0,087.

The information ratio graph helps us to understand better if the QLib model outperformed and how much the performance are better than the benchmark performance. In Figure 20 the information ratio for excess return series with and without cost has been displayed:

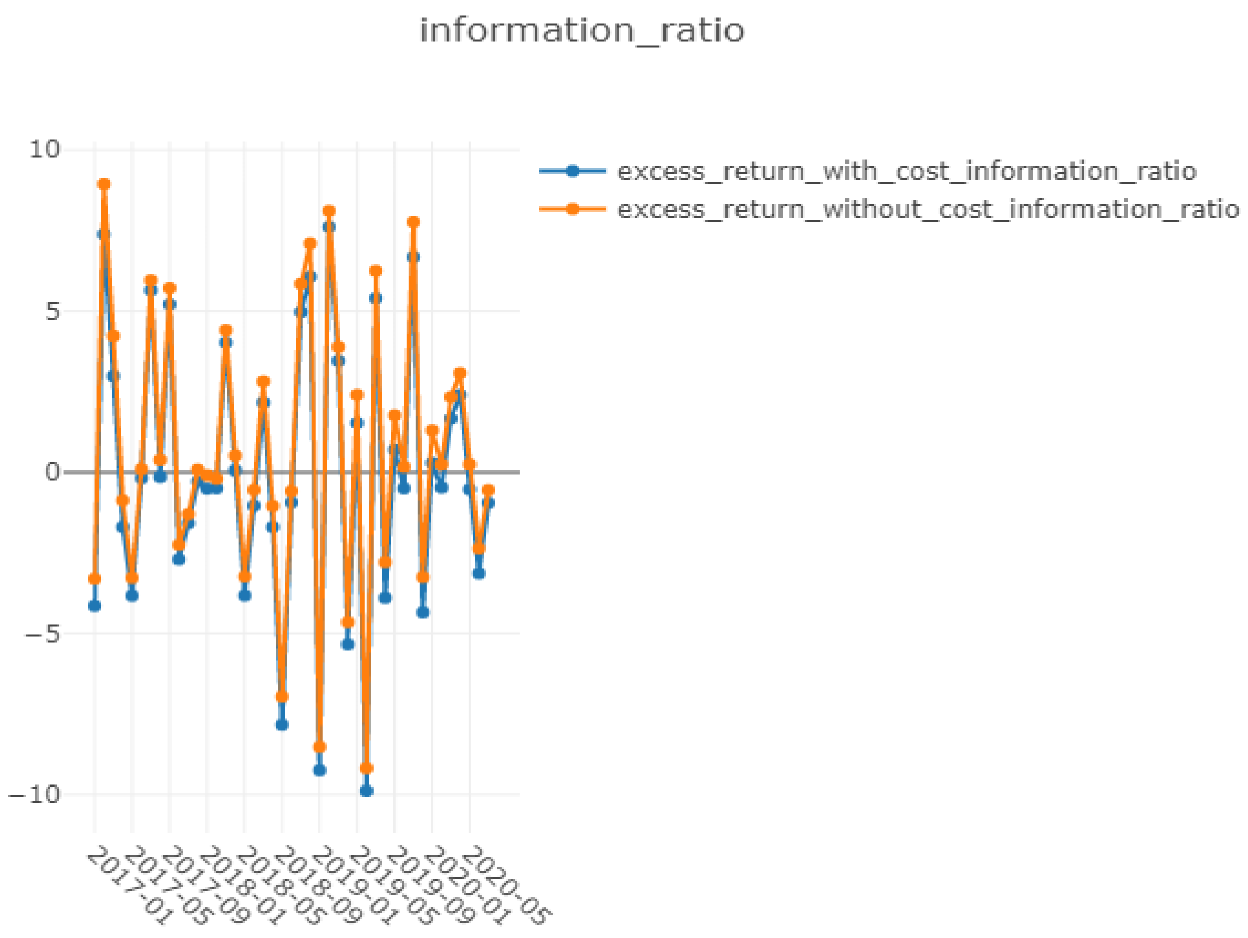


Figure 20: information ratio trend between Qlib LSTM models and benchmark

Unlike the LSTM model information ratio, the Qlib LSTM model information ratio trend doesn’t show a constant positive trend. This means that the Qlib model can’t outperform constantly the benchmark. If for the excess return with cost portfolio isn’t a surprise, given that the information ratio showed in the numerical results is negative, about the excess return without cost portfolio is more astonishing. The information ratio is equal to 0,57, but the information ratio trend compared to the benchmark is far from being positive.

To study the volatility of the portfolio, the standard deviation series has been reported in Figure 21:

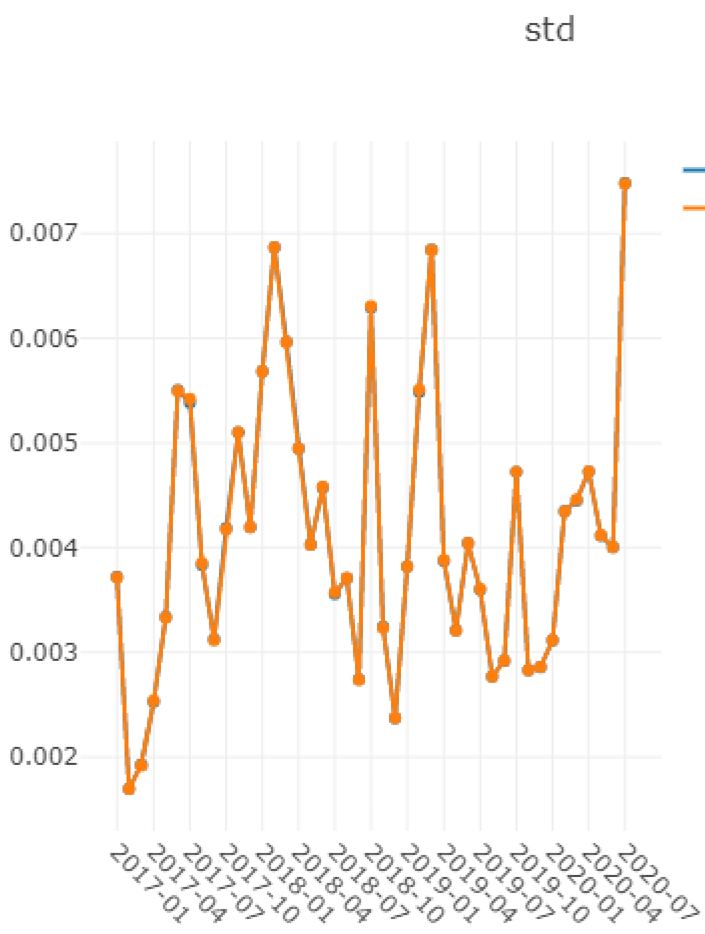


Figure 21: standard deviation time series Qlib LSTM model

As shown in this figure, we can see that the portfolios are affected by higher volatility than the LSTM model. A higher volatility means that the portfolio's value can change seriously over a short period of time in both negative and positive direction. Usually investors tend to fear a high volatility because it means a higher risk for their investment.

Finally, the cumulative return graph has been displayed in Figure 22:

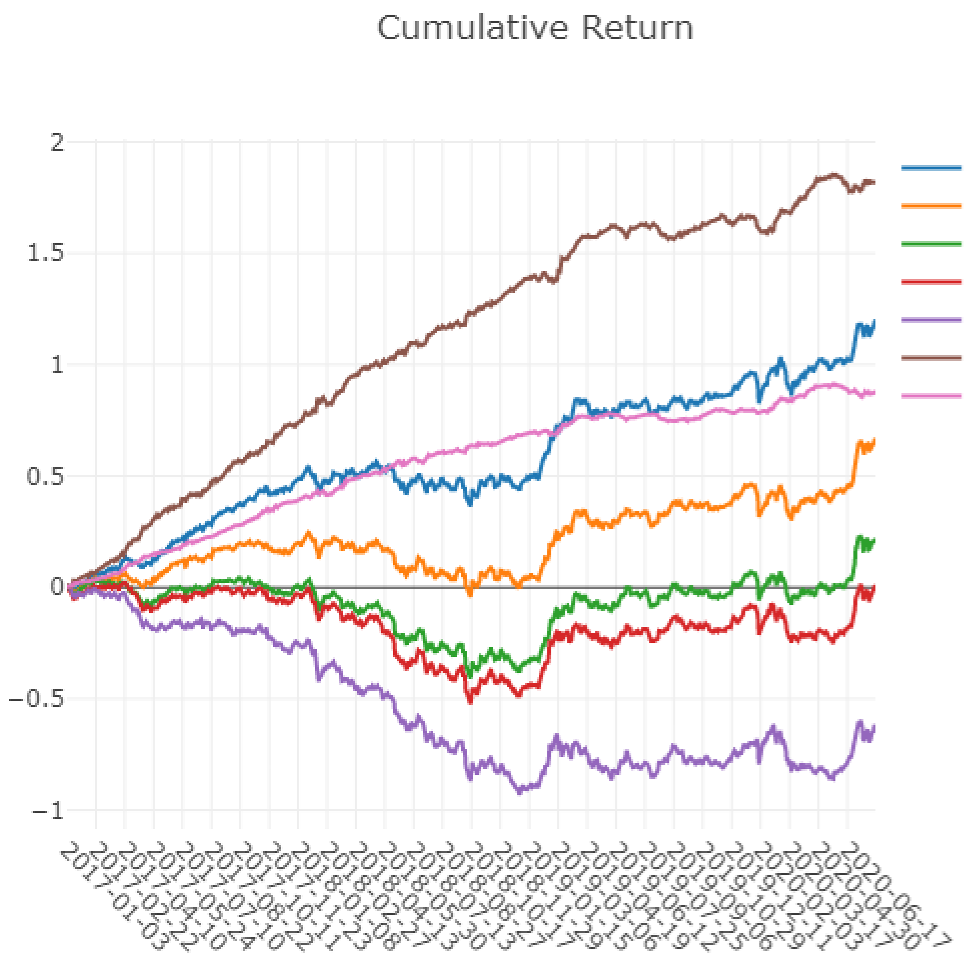


Figure 22: Qlib LSTM model performance on cumulative return series

Despite the investment portfolio has always registered a positive trend in both long-short and long-average time series, just two of the groups have always listed a positive trend during the test. Nevertheless, the long-short and long-average series resulted positive, underlying that generally the Qlib LSTM model relatively outperforms the benchmark.

4.4 Models' comparison

The models' backtest results have been split in two tables: Table 1 resumed the backtest results for a portfolio without trading cost, while Table 2 reported the backtest results for a portfolio with trading cost.

	LSTM model woc	Qlib LSTM model woc
Mean	0,000425	0,000161
Standard deviation	0,004399	0,004346
Annualized return	0,101141	0,038221
Information ratio	1,490447	0,570075
Max drawdown	-0,073277	-0,086672

Table 1: backtest results without trading cost (woc)

	LSTM model wc	Qlib LSTM model wc
Mean	0,000258	-0,000004
Standard deviation	0,004396	0,004344
Annualized return	0,061292	-0,000935
Information ratio	0,903698	-0,013957
Max drawdown	-0,095516	-0,098130

Table 2: backtest results with trading cost (wc)

Comparing the performance's models, it appears clear that the LSTM model based on Qlib financial dataset performs better than the LSTM model built on Qlib financial data enriched with the sentiment feature. Despite the LSTM model has a higher standard deviation (so a higher volatility) than the Qlib model, the base LSTM model has a higher mean, annualized return, information ratio and a lower maximum drawdown than the Qlib LSTM model in both with and without cost market conditions. More in detail, we can see that the Qlib LSTM model in case of presence of trading cost doesn’t permit either a positive annualized return, and it means that the portfolio investment with trading cost can cause a loss for investors.

Despite the Qlib LSTM model fit well the market trend, from these results we may assume that the add of sentiment feature didn’t bring an increase of portfolio’s value. The cause of these results can be multiple:

- Overfitting: the model might be too complex and therefore, might overfit the training data and not generalize well to new unseen data.
- Curse of dimensionality: the curse of dimensionality is a phenomenon in machine learning and statistics that refers to the difficulty of accurately modeling high-dimensional data. This is because the number of observations required to estimate a model's parameters grows exponentially with the number of dimensions in the data. As the number of features increases, the amount of data required to effectively train the model also increases. With limited data, the model might not be able to effectively learn the relationship between the features and the target variable, leading to poor results.
- Noise: a large number of features can introduce noise into the model, making it more difficult for the model to distinguish the important features from the irrelevant ones.
- Computational Complexity: LSTM models can be computationally expensive, especially as the number of features increases. With a large

number of features, the model might take too long to train, or require a large amount of computational resources.

- Feature engineering and selection: the choice of Tweets used can add noise in the data, leading to overfitted and decreased accuracy. As we can see, the Qlib LSTM model has a higher Mean Squared Error, that means the model doesn't fit better than the base LSTM model.

Following these results, we can assume that a sentiment feature based on the tweets of these ten accounts didn't bring added value to the portfolio investment strategy. In fact, it resulted to be harmful to predict an optimal investment portfolio. We need to remember that the reliability of tweet content can vary greatly and is dependent on a number of factors, because they can be written by anyone and can contain inaccuracies, opinions, or even outright falsehoods. Additionally, tweets can be easily manipulated or spread false information through networks, underlying the importance of understanding the source and credibility of the information being shared. Despite we've chosen ten precise Twitter accounts, we still don't know how much the content of every tweet is reliable. When we talk about tweet content unreliable we are talking about fake news. Fake news refers to deliberately false or misleading information that is spread through various media, such as traditional news outlets, social media, or online news websites. The spread of fake news can have serious consequences, including the dissemination of false information, the manipulation of public opinion, and the exacerbation of societal divisions. In today's world, where information is readily available and can be shared with millions of people at the click of a button, the problem of fake news has become more pronounced. Given that, in this dissertation we demonstrated that the tweet content of these 10 accounts couldn't be used as a predictive feature for quantum investment portfolio building. Moreover, it's crucial to remind that the results obtained throughout this dissertation can’t be replicate in different environments with different key parameters like market region, period of time, Twitter accounts and algorithms exploited.

Conclusion

In conclusion, this dissertation explored one approach in the investment strategies related to the use of sentiment analysis on social media contents in stock market prediction. We compared two LSTM models trained on Chinese stock market data retrieved on Yahoo finance. We aimed to evaluate the performance of an LSTM model built exploiting the Qlib repository developed by Microsoft, and a modified LSTM model that included a sentiment feature calculated using a version of RoBerta on tweets related to general news. The investment strategy implemented has been the TopKDropout strategy, that permitted to the model to change a preselected number of stocks in order to improve the portfolio's performance. The first LSTM model was trained on the basic dataset including the financial indicators, while the second LSTM model was trained on an enriched dataset included the sentiment feature, calculated on tweets collected by ten Twitter accounts from 2008 to 2020. The results showed that the LSTM model trained on the basic dataset outperformed better than the LSTM model trained on the enriched dataset. The investment portfolio without cost of the first LSTM gained an annualized return equal to 10,11%, while the investment portfolio of the second LSTM model reached an annualized return equal to 3,82%. The poor performance of the LSTM trained on the enriched dataset could be because we've used a precise combination of parameters, such as the Chinese market region, time horizon, or Twitter accounts. For future developments, it could be interesting to test the performance of the LSTM model trained on a new enriched dataset collected from different market regions or on a different time horizon. Moreover, it may be significant test the performance of a new dataset enriched with a feature sentiment obtained using a different combination of Twitter accounts. This could help to validate the hypothesis that the poor performance of the LSTM model with the enriched dataset was due to the specific combination of parameters used in this study. Overall, this dissertation highlights the importance of considering multiple factors when building and evaluating LSTM models and provides valuable insights for future research on sentiment analysis related to quantum investment field.

Bibliography

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv*. doi:10.48550/ARXIV.1409.0473

Bing, L. (2012). *Sentiment Analysis and Opinion Mining.* Springer Cham. doi:https://doi.org/10.1007/978-3-031-02145-9

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*. doi:10.48550/ARXIV.1810.04805

Face, H. (2016). *Twitter roberta base sentiment latest*. Retrieved from https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest

Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning.* Cambridge, MA, USA: MIT Press. Retrieved from http://www.deeplearningbook.org

Hochreiter, S., & Schmidhuber, J. (1997, 12). Long Short-term Memory. *Neural Computation, 9*, 1735-80. doi:10.1162/neco.1997.9.8.1735

Hull, J. (2018). *Risk management and financial istitutions.* Wiley.

JustAnotherArchivist. (2018). Retrieved from snscrape: A social networking service scraper in python: https://github.com/JustAnotherArchivist/snscrape

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., . . . Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv*. doi:10.48550/ARXIV.1907.11692

Loureiro, D., Barbieri, F., Neves, L., Anke, L., & Camacho-Collados, J. (2022). TimeLMs: Diachronic Language Models from Twitter. *arXiv*. doi:10.48550/ARXIV.2202.03829

Medium. (2020). *Recurrent neural network and its variants*. Retrieved from https://medium.com/analytics-vidhya/recurrent-neural-network-and-its-variants-de75f9ee063

Microsoft. (2020). Retrieved from Strategy: https://qlib.readthedocs.io/en/latest/component/strategy.html

Microsoft. (2020). Retrieved from Component data: https://qlib.readthedocs.io/en/latest/component/data.html

Microsoft. (2020). *Github*. Retrieved from https://github.com/microsoft/qlib

Microsoft. (2020). *Introduction*. Retrieved from https://qlib.readthedocs.io/en/latest/introduction/introduction.html

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024--8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Richardson, L. (2007). Retrieved from Beautiful soup documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

The pandas development team. (2020, feb). pandas-dev/pandas: Pandas. *Zenodo*. doi:10.5281/zenodo.3509134

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., . . . Polosukhin, I. (2017). Attention Is All You Need. *arXiv*. doi:10.48550/ARXIV.1706.03762

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., . . . Le Scao, T. (2020, 10). Retrieved from Transformers: State-of-the-Art Natural Language Processing: https://github.com/huggingface/transformers

Yang, X., Liu, W., Zhou, D., Bian, J., & Liu, T.-Y. (2020). Qlib: An AI-oriented Quantitative Investment Platform. *arXiv*. doi:10.48550/ARXIV.2009.11189