جامعة خليفة
Khalifa University

Stock Ranking Analysis using AI

Saeed Mohamed Bark AlMarri

MSc. Thesis

May 2023

A thesis submitted to Khalifa University of Science and Technology in accordance with the requirements of the degree of M.Sc. in Computational Data Science in the Department of Electrical Engineering and Computer Science.

Stock Ranking Analysis using AI

By

Saeed Mohamed Bark AlMarri

A thesis submitted in partial fulfillment of the requirements for the degree of

M.Sc. in Computational Data Science

at

Khalifa University

Thesis Committee

Dr. Panagiotis Liatsis (Main Advisor),
Khalifa University
Dr. Zeyar Aung (Co-Advisor)
Khalifa University

Dr. Mohamed Riahi (RSC Member 1),
Khalifa University
Dr. Deepak Puthal (RSC Member 2)
Khalifa University

May 2023

Abstract

Saeed Bark AlMarri, "**Stock Ranking Analysis using AI**", M.Sc. Thesis, MSc in Computational Data Science, Electrical Engineering and Computer Science, Khalifa University of Science and Technology, United Arab Emirates, May 2023.

Financial markets are embedded and have had significant impact on many areas such as business, jobs, education and technology, which all have an impact on the economy. It is an extremely challenging field to accurately predict because stock market movements and price behavior are hard to analyze given its dynamic, nonstationary, nonparametric, nonlinear, and chaotic nature and they are impacted by several interrelated factors such as economic, political, psychological, and company-specific variables [1]. There is a rich history of stock prediction using analytical and traditional statistical tools such as Moving Average, Auto-regressive Integrated Moving Average, and Vector Auto-regressive Moving Average. But recently, we have the emergence of advanced Machine Learning methods that provide an entirely new toolbox that could be used to address complex prediction problems.

This research will focus on developing Predictive Equity Ranking (PER) methodologies using modern machine learning tools and Learning to Rank (LTR) algorithms. The dataset that would be used is a real-world dataset (S&P 500). The expected outcome is a stock ranking based on the most profitable stocks. By succeeding in doing so, a portfolio can be further constructed with the highest revenue generating stocks, for both long and short strategies.

Indexing Terms: LTR, Learning to Rank, Stocks Ranking, Predictive Equity Ranking, Listwise

Acknowledgement

Before beginning the acknowledgment, I would like to express my utmost gratitude to the leadership of my country, the United Arab Emirates, for their unwavering support and encouragement towards learners and individuals eager to continue their academic journey. Then,

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Panagiotis Liatsis, whose unwavering guidance, dedication, expertise, and patience have been instrumental in shaping this thesis. I am as well thankful for my co-advisor, Dr. Zeyar Aung, whose invaluable support, accessibility, insights, and encouragement greatly contributed to this work.

I would also like to extend my heartfelt appreciation to my employer, Abu Dhabi Investment Authority, and its visionary leadership, who provided me with the opportunity to pursue my master's degree and acquire knowledge that will undoubtedly contribute to the ongoing digital transformation in the UAE.

My profound thanks go to my family members, whose patience, understanding, and support during my master's journey were a constant source of strength and motivation. Their unwavering belief in my abilities and celebration of my achievements provided me with the inspiration to persevere.

Furthermore, I am grateful to my ADIA colleagues and classmates for sharing this enriching experience, navigating challenges, and creating lasting memories.

This thesis represents the collective effort and sacrifices made by all, marking a successful conclusion and the beginning of a journey towards greater contributions.

"You are the average of the five people you spend the most time with" - Jim Rohn

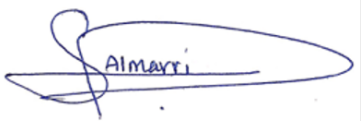
Declaration and Copyright

Declaration

I declare that the work in this thesis was carried out in accordance with the regulations of Khalifa University of Science and Technology. The work is entirely my own except where indicated by special reference in the text. Any views expressed in the thesis are those of the author and in no way represent those of Khalifa University of Science and Technology. No part of the thesis has been presented to any other university for any degree.

Author Name: Saeed Bark AlMarri

Author Signature:



Date: May 10, 2023

Copyright ©

Table of Contents

Abstract I

Acknowledgement II

Declaration and Copyright III

Table of Contents IV

List of Figures VII

List of Abbreviations VIII

1 Introduction 1

1.1 Forecasting Financial Markets 3

1.1.1 Price prediction 3

1.1.2 Learning to Rank 3

1.2 Problem Statement 5

1.3 Aim and Objectives 5

1.4 Thesis Organization 7

2 Literature Review 8

2.1 Overview of Stock Forecasting 8

2.2 LTR Applications 9

2.2.1 LTR in Information Retrieval 10

2.2.2 LTR in Financial Markets 14

3 Dataset and Data Preprocessing **22**

3.1 Data Collection 22

3.2 Data Characteristics 22

3.3 Data Preprocessing 24

3.4 Feature Generation and Selection 26

3.4.1 Importance of Feature Generation 26

3.4.2 Feature Selection 26

3.4.3 Datetime Features 29

3.4.4 Moving Averages 29

3.5 Final Dataset 30

3.6 Model's Input vs Output 31

3.7 Summary 32

4 Proposed Model **33**

4.1 Gradient Boosting Machine 33

4.1.1 What is GBM? 33

4.1.2 Architecture of a Gradient Boosting Machine (GBM) 33

4.1.3 How does it operate? 34

4.1.4 Why GBM? 36

4.1.5 GBM strengths 37

4.2 Learning to Rank 39

4.2.1 Pointwise approach 40

4.2.2 Pairwise approach 41

4.2.3 Listwise approach 42

4.2.4 Summary 43

5 Results **44**

5.1 Regress-then-Rank (Root Mean Square Error (RMSE)) 44

5.2 Learning-to-Rank Pairwise 45

5.3 Learning-to-Rank Listwise 46
5.3.1 QueryRMSE 46
5.3.2 YetiRank 46
5.3.3 YetiRankPairwise - Hybrid Approach 47
5.4 Performance Comparison 48
5.4.1 Metrics 48
5.4.1.1 NDCG 48
5.4.1.2 MAP 49
5.4.1.3 RBO 50
5.4.2 Performance Comparison: Different Investment Horizons 51
5.4.2.1 Monthly Ranking 51
5.4.2.2 Quarterly Ranking 52
5.4.2.3 Yearly Ranking 53
5.5 Summary 54

6 Conclusion 56
6.1 Future Work 57

References 58

Appendices 65
Implementation 65

List of Figures

3.1 Collected Dataset for S&P 500 stocks 23
3.2 Collected Dataset Description 24
3.3 Stock Existence Timeline 25
3.4 Final Dataset 30
3.5 Final Dataset Description 31

4.1 How Gradient Boosting Decision Tree operates 36
4.2 Comparison with baselines: log loss / zero-one loss (relative increase for baselines). 37
4.3 LTR framework 39
4.4 Complexity-Performance Trade off between three methods of LTR 40
4.5 Comparison between three methods of LTR 40

List of Abbreviations

ML Machine Learning

NN Neural Network

DL Deep Learning

DNN Deep Neural Network

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

RMSE Root Mean Square Error

DMTL Deep Multi-Task Learning

MLP Multi Layer Perceptron

PER Predictive Equity Ranking

LTR Learning to Rank

GBM Gradient Boosting Machine

NDCG Normalized Discounted Cumulative Gain

MAP Mean Average Precision

RBO Rank Biased Overlap

Chapter 1

Introduction

The stock market is one of the primary indicators of a country's economic strength and development. A stock market is a public market that lists companies' shares and allows ordinary investors to take a share representing ownership of the company they invest in, equivalent to the price paid. Stock exchanges are trusted third parties where one can buy and sell shares. A stock price depends on the demand and supply of the stock. The price will go up if the demand for the stock is high and vice versa. By allowing businesses to trade publicly, companies can use the stock market to raise money and expand capital. A rising share price will increase the business investments and growth of the company's profitability. Share prices will affect the wealth of households and their consumption. Therefore, central banks tend to keep an eye on the control and behavior of the stock markets [2].

Traditional financial prediction approaches are separated into two groups: technical analysis and fundamental analysis. Fundamental analysis is based on internal and external factors regarding a company. Interest rates and exchange rates are the main external factors to be considered, while companies' press releases and balance sheet disclosures are examples of internal factors used in prediction processes. On the other hand, technical analysis utilizes past price data and technical indicators to predict future behavior of financial time series. Therefore, stock returns have notoriously been difficult for analysts to predict accurately due to their fluctuations being of unclear origins.

The strong growth of interest in the stock market over the past two decades is no surprise, given that assets worth billions of dollars are traded on stock exchanges every day, and investors want to maximize their profits by timing their actions on the market. On the other hand, researchers from various fields also want their share of the pie. For this purpose, a research field has developed where both investors and researchers test models for the relationship between the historical behavior of stock prices and their future movements.

Developments in the field of artificial intelligence, specifically Machine Learning, create opportunities for their use in financial prediction tasks. GBM are a popular machine learning algorithm used in the development of learning-to-rank models in financial markets. Learning-to-rank is the task of ranking a set of items, such as stocks or investment opportunities, based on their relevance to a specific query or user preference.

GBMs work by iteratively improving the accuracy of a weak learner, such as a decision tree, by fitting the residual errors of the previous iteration. The result is a strong learner that can accurately rank items based on their features and characteristics.

In financial markets, learning-to-rank models can be used to analyze and rank stocks based on their historical performance, financial ratios, and other relevant data. This information can then be used to make informed investment decisions and optimize portfolio performance.

GBMs have several advantages in the development of learning-to-rank models in financial markets, including their ability to handle complex and non-linear relationships between variables, their robustness to outliers and noise in the data, and their ability to model interactions between variables.

However, it is important to note that GBMs, like any machine learning algorithm, require careful tuning and validation to avoid overfitting and ensure reliable results. It is also important to consider the potential biases and limitations of the data used to train the model and to interpret the results in the context of the specific financial market and investment strategy being considered.

1.1 Forecasting Financial Markets

Forecasting financial markets is inherently difficult due to the unpredictable nature of financial markets and the multitude of factors that can impact stock prices. Traditional methods of financial market prediction, such as technical analysis and fundamental analysis, have been used for many years. However, these methods are limited in their ability to accurately predict the stock market and often rely on subjective human interpretation of data. AI-based methods have emerged as powerful tools for forecasting financial markets, as they are able to analyze vast amounts of data and detect patterns and trends that may not be apparent to human analysts [3]. Moreover, some studies have used multiple data sources, such as news, financial reports, and social media, to improve the accuracy of financial market predictions [4].

1.1.1 Price prediction

Predicting stock prices accurately is essential for investors and financial institutions. However, traditional methods of stock price prediction, such as technical and fundamental analysis, are often subjective and do not account for the complex non-linear relationships between various factors that affect stock prices. In recent years, artificial intelligence (AI) has emerged as a promising tool for stock price prediction, as it can analyze large volumes of data and detect patterns and trends that may be difficult for human analysts to discern [3]. Studies have shown that AI-based methods can provide accurate and robust predictions of stock prices, demonstrating their potential for practical use.

1.1.2 Learning to Rank

LTR is a machine learning technique used in search engines and other applications to determine the most relevant results for a user's query. LTR teaches a computer to rank items based on their relevance to a given query, much like how a human would rank them. The ranking can be based on factors such as keyword frequency, document age, popularity, or user behavior and

can be tailored to specific tasks or domains like web search, e-commerce, multimedia, or social networks [5].

LTR is an important field in information retrieval and can significantly improve the quality of search results and user satisfaction. For example, using LTR, a search engine can show the most relevant and diverse results on the first page instead of burying them under multiple pages. Similarly, e-commerce websites can recommend products based on user behavior and preferences instead of displaying a generic list [6]. LTR can also enable new applications and services, like personalized news feeds, question-answering systems, or voice assistants that require intelligent and adaptive ranking [7].

LTR encompasses a wide range of models and methods, from simple linear models to complex neural networks, that aim to learn the optimal ranking function from training data. These methods can be classified into pointwise, pairwise, and listwise approaches, depending on how they transform the input-output pairs [4]. Pointwise methods treat each item-query pair as an independent instance and predict its relevance score directly. Pairwise methods compare two items with respect to a query and predict their relative order. Listwise methods consider the entire list of items for a query and optimize a ranking objective, such as Discounted Cumulative Gain (DCG) or Mean Average Precision (MAP). Despite the technical complexity, LTR has many practical applications and benefits for both users and businesses.

LTR can be used to predict stock prices, forecast market trends, and identify potential investment opportunities. For example, LTR can be used to rank a set of stocks based on their likelihood to outperform the market, based on various factors such as market volatility, company financials, and industry trends.

Moreover, LTR has been applied to stock market prediction by ranking stocks based on their expected performance [3]. In LTR-based stock market prediction, historical data is used to train a ranking model that assigns a score to each stock based on its predicted performance. The model is then used to rank the stocks, with the top-ranked stocks being expected to perform better than the lower-ranked ones. Several studies have demonstrated the effectiveness of LTR-

based approaches for stock market prediction.

1.2 Problem Statement

Stock forecasting plays a key role in developing stock investment strategies [8] and portfolios. An investor has two main decisions to make regarding their stock portfolio: buy or sell. They would want to sell the least performing stocks and buy the best-performing ones. Moreover, there must be priorities for some stocks over others for capital allocation purposes, which require a ranking of those stocks. To achieve this goal, the solution for stock ranking is equivalent to the Predictive Equity Ranking (PER).

PER is a concept that combines predictive analytics, data-driven insights, and various financial metrics to rank and evaluate the potential performance of public and private companies. The aim is to provide investors, analysts, and other stakeholders with a data-backed perspective on the future performance of these companies, enabling them to make more informed decisions when it comes to investment and resource allocation. PER involves using various information sources to generate stock rankings and obtain the highest possible returns, and it is often performed using machine learning strategies. Due to the relatively new concept of PER, the financial field is in need of better techniques for developing stock portfolio selection strategies. This is an open problem that requires exploration of new and advanced techniques, which would lead to better decision making and more investment revenue.

1.3 Aim and Objectives

The aim of this thesis is to develop and evaluate a machine learning model for stock ranking prediction. The objectives of this study are:

- To review the literature on machine learning methods for stock market prediction, with a focus on LTR algorithms and the use of machine learning tools such as Gradient Boosting

Machine GBM. GBM unites numerous weak decision tree learners into a potent ensemble, making it a versatile machine learning algorithm for diverse applications such as classification, regression, and ranking problems.

- To develop a predictive equity ranking model using GBMs for stock returns prediction and learning to rank.
- To evaluate the performance of the proposed model on real-world S&P 500 financial datasets.
- To compare the proposed model with traditional machine learning methods and baseline models.
- To analyze the factors affecting the performance of the model and identify potential areas for improvement.

1.4 Thesis Organization

This thesis is organized as follows:

- Chapter 2 provides a Literature Review on GBMs, LTR algorithms, and their application in financial markets.
- Chapter 3 discusses the Dataset and Data Preprocessing, including the S&P 500 dataset, preprocessing techniques, feature generation, and selection.
- Chapter 4 presents the Proposed Model, detailing the development of the predictive equity ranking model using GBMs for stock returns prediction and LTR algorithms for learning the correct rank based on next month, quarter, and year returns.
- Chapter 5 showcases the Results, which include the experimental results and performance analysis of the proposed model. It also compares the model's performance with traditional machine learning methods and baseline models.
- Chapter 6 concludes the thesis with a Conclusion, summarizing the key findings and their implications. It also discusses the contributions of the study, its limitations, and suggests future directions for research.

Chapter 2

Literature Review

2.1 Overview of Stock Forecasting

Stock forecasting is a widely researched field within financial markets and has attracted considerable attention from both academia and industry. The primary goal of stock forecasting is to predict the future price or movement of stocks to facilitate informed investment decisions. Accurate forecasting models can potentially lead to significant financial gains, making it an essential aspect of financial market analysis.

Several methods and techniques have been proposed in the literature to address the stock forecasting problem. These methods can be broadly categorized into statistical methods, technical analysis, and Machine Learning (ML) methods.

ML methods have emerged as a promising approach to stock forecasting due to their ability to learn complex relationships and adapt to new data. These methods can be broadly classified into supervised learning, unsupervised learning, and reinforcement learning techniques. Supervised learning methods, such as Support Vector Machines (SVM) [9], Artificial Neural Networks (ANN) [10], and Random Forest [11], have been applied to predict stock prices or classify stock movements. More recently, reinforcement learning algorithms, such as Q-Learning [12], have been employed to optimize trading strategies based on stock forecasts.

In recent years, Deep Learning (DL) techniques have gained popularity in stock forecast-

ing due to their ability to model complex and nonlinear relationships in data. Methods such as Recurrent Neural Network (RNN) [13], Long Short-Term Memory (LSTM) [14], and Convolutional Neural Networks (CNN) [15] have demonstrated promising results in predicting stock prices and movements.

Despite the advancements in stock forecasting methods, accurately predicting stock prices remains a challenging task due to the inherent uncertainty and dynamic nature of financial markets. Researchers continue to explore novel techniques and approaches to improve the accuracy and robustness of stock forecasting models.

2.2 LTR Applications

Learning to rank algorithms have been applied in various fields, including stock prediction. These algorithms are designed to learn from historical data and rank a set of items based on their relevance to a given query. In the context of stock prediction, learning to rank algorithms aim to predict future stock prices by ranking a set of stocks based on their likelihood to increase or decrease in value. These algorithms typically use a combination of features such as technical indicators and financial ratios, and machine learning models, such as neural networks and GBMs, to generate predictions.

The effectiveness of learning to rank algorithms in stock prediction has been demonstrated in several studies, with promising results in terms of accuracy and profitability. Several studies have explored the application of LTR algorithms in stock prediction. LTR algorithms are a type of machine learning method that are commonly used in information retrieval tasks, such as web search and recommendation systems. These algorithms have been found to be effective in ranking stocks based on their expected performance.

The review examined several LTR algorithms, including pointwise, pairwise, and listwise approaches, and compared their performance using various evaluation metrics such as Normalized Discounted Cumulative Gain (NDCG), Mean Average Precision (MAP), Rank Biased Over-

lap (RBO), and Precision. Overall, the review found that pointwise LTR algorithms have many drawbacks and, most importantly, are not suitable for the complexity of stock ranking and therefore yield significantly lower accuracy scores. On the other hand, pairwise and listwise LTR algorithms are more suitable for stock ranking and have the potential to improve the accuracy and efficiency of stock prediction models, but more research is needed to overcome the challenges associated with data quality and feature selection.

2.2.1 LTR in Information Retrieval

Learning to Rank (LTR) is a machine learning technique widely used in Information Retrieval (IR) applications. LTR enables systems such as search engines, e-commerce sites, and recommendation systems to provide users with more accurate, relevant, and personalized results. By analyzing various features such as keyword match, document popularity, and user behavior, LTR algorithms can rank documents based on their relevance to a user's query. LTR has become an essential tool in many IR applications, enhancing the overall user experience by providing more useful and accurate results.

Keshvari et al. [16] introduced ListMAP, a novel technique for Information Processing and Management by learning-to-rank (LTR) algorithms, that uses a unique ranking model based on a distribution that encodes training data and assigns weights to instances in that data. The ListMAP loss function is formalized as a posterior estimation problem, implemented using the Encoder part of the Transformer neural network. The authors evaluated the performance of ListMAP on various datasets, including MQ2007 and MQ2008 from LETOR 4.0 benchmark, which feature collections of L2R models, such as YahooSet1!, YahooSet2!, Microsoft 30k, and Microsoft 10k. The evaluation metrics used were Normalized Discounted Cumulative Gain (NDGC) and Mean Reciprocal Rank (MRR) at various positions, with results showing that ListMAP outperforms existing LTR algorithms.

Recent advances in deep learning have shown significant improvements in recommendation systems. One such approach is DeepRank, a listwise deep learning framework for collaborative

filtering proposed by Chen and Zhou [17]. DeepRank is designed to capture latent features of users and items using a nonlinear model architecture, which provides insights into collaborative filtering. The system is based on a listwise ranking strategy, which ensures better accuracy and speed than previous models. To evaluate its performance, DeepRank was tested on three different public experiments: MovieLens100k, MovieLens1M, and Yahoo! Movie. The model's loss function was based on matrix factorization, and the performance was evaluated using RQ1, RQ2, and RQ3 metrics. The results show that DeepRank outperforms previous models in terms of accuracy and speed, and has the potential to provide new insights into collaborative filtering. The success of DeepRank highlights the potential of deep learning-based approaches for building recommendation systems in information retrieval.

In the field of information retrieval, the implementation of learning to rank (LTR) techniques has been widely studied to improve the effectiveness of information retrieval systems. One approach is the use of SoftMax-based approximation to derive a ranking function in a listwise algorithm [18]. Thibaut Thonet et al. introduced a model design that incorporates metrics such as Precision@K, Mean Average Precision (MAP), and NDCG to obtain the rank and sort functions. This approach showed efficiency in terms of four features, including SmoothI, quality of approximation, application to IR metric, and gradient stabilization in neural architectures. The dataset used in this model is LETOR 4.0, which includes MQ2007, MQ2008, and MSLR-Web30K. The SmoothI component proved to be a key differentiator in the analysis of the model compared to previous models [19].

In similar manner, the article [20] "Comparing Pointwise and Listwise Objective Functions" by Muhammad Ibrahim and Mark Carman, proposes a ranking solution to the problem in a pointwise approach, using random forest and learning-to-ranking algorithms for classification and regression framing. The aim of the model is to optimize the NDCG metric in a greedy manner. However, direct optimization of listwise objective functions is computationally challenging for most learning algorithms. To achieve the purpose, the implementation is broken down into three stages. Firstly, an unsupervised partitioning strategy is employed to assess the effective-

ness of the random split. Secondly, the pointwise and listwise approaches are utilized to address the data overfitting problem. Finally, the predictive accuracy is evaluated by constructing tree structures of all algorithms. The model is tested on various datasets, and the most widely used dataset, MQ2007, contains 1,000 queries (45,000 query-document pairs) in the training set.

The exploration-exploitation dilemma in online learning to rank retrieval can be a challenge, but a possible solution is to use either pairwise or listwise learning methods. Researchers Katja Hofman, Shimon Whiteson, and Maarten de Rijke conducted an experiment using LETOR 2.0 and LETOR 3.0, two standards of learning to rank models featuring nine datasets extracted from TREC and GOV2. Their findings showed that feedback plays a crucial role in achieving accurate results, as noisy feedback can lead to different outcomes. The researchers used cumulative NDCG to measure the system's performance, and ran all experiments for 1000 iterations. For pairwise learning, they evaluated offline performance for g values of 0.0001, 0.001, 0.01, 0.1, and selected the best setting (g = 0.001) for all data sets. Meanwhile, for listwise learning, they used the best performing parameter settings , which were d = 1 and a = 0.01 and resulted in good performance across all datasets in their preliminary experiments. [21]

Moreover, a proposed List Rank-MF approach for collective filtering using a listwise learning-to-rank algorithm with Matrix Factorization (MF) approach to overcome the issues of traditional collaborative filtering techniques. They utilized the MovieLens dataset consisting of 10,000 ratings ranging from 1 to 5 for a collection of 1682 items. Multiple protocols, including weak generation, were employed to evaluate the proposed approach. The results showed that the List Rank-MF algorithm achieved higher accuracy compared to the Cofirank-algorithm in most cases [22]. The loss function used in this approach was based on the distance between the ranked and output list, and probabilistic matrix factorization was utilized. The performance of the system was measured using the NDCG metric. The comparison between List Rank and Cofirank algorithms demonstrated that List Rank achieved a 15% success rate, while Cofirank achieved only a 5% success rate [22].

In the field of information retrieval, Listwise LTR algorithms are designed to overcome is-

sues encountered in previous models. [23] proposes a new loss function based on four models: uRank, uBoost, uMart, and urBoost. Effective matrix calculations were implemented by the authors, Xiaofeng Zhu and Diego Klabjan, and multiclass classification was applied in each class to choose the highest-ranking document from the dataset. The four datasets used for training and validation were OSHUMED, MQ2007, Microsoft 10k, and Microsoft 30. The evaluation metrics used in this study were NDCG, ERR, and some functions adapted from Vanilla RNN. The most significant methods implemented in this model were neural networks, neural networks with gradient boosting, and regression trees with gradient boosting. The results show that the proposed model is more effective than traditional models, demonstrating the potential of the Listwise LTR approach [23].

Table 2.1: LTR in IR Literature Review Summary

Citation	Approaches	Model	Dataset
[24]	Learning to rank informational retrieval	Influence Quantification Module Influence Allocation Module	Labeled dataset from analyst reports
[25]	Presentation bias, Learning to rank, counterfactual inference	Rank-based IR metrics Generative click models	Implicit feedback data
[26]	Collaborative filtering social relationship learning to rank	LTRW ResSSN	Stock database from news sources

[27]	Machine learning-based approach Viral Node Identification	Neighborhood-based methods	Relational Graphs from Wikidata.

2.2.2 LTR in Financial Markets

[28] Saha supports the study with a focus on the stock price. The author has mentioned that predicting the stock price movement can be highly challenging because of its volatile nature. The author has argued that based on Efficient Market Hypothesis (EMH), estimating stock price may not be economically viable because the current stock price only reflects the existing information in the "efficient market." The paper is based on the LSTM model to evaluate various existing evaluation measures for "stock ranking performance production." After the analysis, the author has argued that there are some setbacks for all evaluation measures [28]. For instance, MRRT is only focused on the topmost stock, and there can be significant changes in the cumulative investment return if the top two stocks are swapped.

In addition to the LSTM model, the paper used the temporal embedding layer. Using the graph-based techniques and optimal loss function, the author proposed a new performance evaluation measure known as NRBO@k and used the node embedding techniques to perform stock ranking prediction effectively. The study has three crucial findings. Firstly, the author found that list-wise loss leads to better NRBO@10 in almost 3 out of 4 cases with some embedding techniques. From the findings, the paper shows that NRBO@10 enhances the 2.65% while the list-wise loss is used for comparing the combined pairwise and point-wise loss for NASDAQ using wiki data graphs. Secondly, the study shows that the list-wise loss function is a better choice over the combination of pair-wise and point-wise functions.

Lastly, the author has mentioned that influencer and market investor sentiment is used in behavioural finance studies. Most importantly, the primary financial data vendors such as Thomson Reuters and Bloomberg take into account the investor's sentiments in a real-time fashion. Based on the enhanced new feed delivery, most recent studies have focused on integrating news

articles with investor sentiment through machine learning and text mining techniques. The author has used the learning-to-rank algorithm and machine learning ranking method to develop quality portfolios on news sentiments. Even though different machine learning algorithms can be used for trading strategy development and financial market prediction, these efforts mostly focus on formulating portfolios based on "return forecasting."

[29] employed machine learning LTR algorithms as a stock portfolio selection approach using two news sentiment indicators. The main useful property for investment applications in learning to rank algorithms is that, unlike traditional algorithms that predict a single output for a single input, these algorithms learn to relatively rank a group of inputs. The primary contribution of this study was to demonstrate that relative-performance based portfolio construction methods, enabled by learning to rank algorithms, can outperform market benchmarks in various environments. Their result was better than the market benchmark, the hedge fund industry performance index, and sentiment-based strategies that do not use learning to rank algorithms.

The work of [30] demonstrated that Neural Network (NN)s are able to extract relevant information directly from raw data and that using raw data instead of technical indicators to train the learning to rank algorithms leads to much better performance. They also showed that score-weighted ranking achieves better performance than equally weighted portfolios. Regarding the architecture, they showed that LSTM architecture performed better than Deep Neural Network (DNN)-based architecture.

[31] introduced a new portfolio selection strategy using learning to rank. The proposed strategy is to long the top 10% of stocks and short the bottom 10%. This strategy adds a constraint to the list-wise loss function that the bottom ranks of the list are just as important as the top ranks. Their results confirmed the advantage of rank prediction over value prediction, and the long-short strategy also consistently achieved better gains than ListMLE.

[28] proposed several novel ideas which proved effective in building a portfolio using learning to rank. The first was the introduction of node embedding. The stock market can be represented by a graph in which the relationship between stocks is depicted. For example, a graph

may depict the relationship between two stocks based on the relationship between stocks, such as whether they are in the same industry. Given that, it was found that the use of graph information can improve the ranking performance or prediction accuracy significantly. A novel metric aimed at the stock ranking prediction task was introduced named Normalized Rank Biased Overlap.

[32]'s work shows that although the list-wise loss function performs the best at stock rank prediction, it has the highest complexity in terms of model learning. To increase the generalization capability and performance of the model, they introduced Deep Multi-Task Learning (DMTL). DMTL acts as a regularization method by forcing the model to share some of its layers to serve other tasks. In their case, the tasks were trend prediction and excess return prediction, and this was proven to yield better results than regularization through complexity reduction that is often used. The proposed architecture used skip connections in addition to attention modules to build the shared encoding layers of the DMTL framework. Their results outperformed several state-of-the-art baseline architectures due to DMTL, which proved to enhance the learning ability of the model, enabling it to tackle difficult tasks such as list-wise stock ranking prediction.

In [33], Poh. et al. present a study that aims to improve the ranking of currencies in the financial market using context-aware Learning to Rank (LTR) models based on top/bottom encoding techniques. They used a dataset of 31 currencies and found that using the LTR approach resulted in a 30% increase in the sharpe ratio, indicating a significant improvement in ranking accuracy.

[33] reassured the superiority of learning to rank methods in comparison to traditional portfolio building strategies and most importantly, the worst performing learning to rank strategy outperformed the best regress-then-rank using DNNs which exhibits the importance of learned ranking. The paper also compared two learning to rank strategies which are pair-wise and list-wise but the results could not determine if one of the methods was better in all situations.

To train the LTR model, the authors of [33] used pairwise logistic and ListNet loss functions

and evaluated the performance using metrics such as Random (Rand), Volatility normalized MACD, and Multilayer perceptron (Multi Layer Perceptron (MLP)). They found that the LTR model was effective in improving the Sharpe ratio in both normal and risk-off market states. Overall, this study highlights the potential of LTR models in improving the ranking of financial assets based on their performance and demonstrates the importance of using context-aware approaches and relevant metrics in training and evaluating LTR models in the financial market.

The paper [34] aims to overcome the issue of financial time series data having multiple possible outcomes (or furcations) by designing a model that can rank stocks based on their expected performance. To develop the model, the authors use a dataset obtained from Borsa Istanbul and combine five different technical indicators (EMA100, RDP-20, RDP-15, RDP-10, and RDP-5) to create a single large dataset for training the SVM. The authors evaluate the performance of the model using a sample set of 1800 features for pairwise validation. The authors use three different loss functions for training the SVM: pairwise, pointwise, and listwise. They found that the pairwise loss function performed the best in terms of ranking accuracy. The proposed model achieved significant success in predicting the direction of stock prices over a specific time period and the magnitude of price changes with respect to the number of days and clients. The paper provides insights into the potential of using SVM-based pairwise techniques to rank stocks based on historical data and overcome the problem of furcation in financial time series data.

Mohammad Alsulmi [35] proposes a rank-based approach using machine learning algorithms for choosing stocks for long-term investment portfolios. The model is based on a unique dataset with multiple features derived from various criteria. The author proposes a new feature statistic that can be used by stock companies. The loss function is chosen based on point-wise, pairwise, and listwise criteria. The author employs a learning-to-rank (LTR) model that includes various algorithms such as Linear Regression, Mart, LambdaMart, LambdaRank, Coordinate Ascent, RankBoost, Random Forests, RankNet, and ListNet. The dataset used for analysis is real-time data from the Saudi stock market. The metrics used for evaluation are NDCG@10 and P@10.

The author concludes that the proposed feature set can be used by companies for investment in the stock market and will help in digitalizing the stock market.

It is crucial to know that investing in the stock and financial market is a challenging task that requires huge efforts from an investment adviser and financial analysis to study the stock exchange and search the investment opportunities[35]. Normally, domain experts are involved in extensive research on stock markets which involves collecting a huge volume of data, including announcements, periodic financial reports, news, etc., to perform different analytical tasks and make investment decisions. With the increase in the data generation capacity of the companies listed in the market and inadequate manual analysis, the financial position of the companies has become difficult, specifically when timely investment decisions are required. The paper has used the LTR model with List Net and List Fold learning for the data analysis in methodology. In addition, the sample for the data analysis has been collected from the market authority of Tadawul [35].

Experts have recently adopted computational intelligence methods for various data analytics applications, which can assist stock market investors and financial analysts in making informed investment decisions and evaluating company capabilities. The literature review aims to analyze stock ranking using a LTR algorithm and explores peer-reviewed papers with different models and rank-based approaches. The literature review focuses on risk-based approaches known as the LTR algorithm. It is important to note that the stock selection task is perceived as a ranking task by nature. Therefore, the literature review aims to evaluate different papers with detailed methodologies to propose solutions to ranking issues. Most of the papers used in this review used data from the China Stock Exchange to understand how rank-based approaches can be used for long-term investment and achieve substantial performance. It is important to mention that LTR is a common strategy used in information retrieval systems to rank search results based on relevance to a query, although LTR can be used in different domains. In this paper, it is only used in the financial domain for ranking stocks based on different criteria such as volatility, returns, and liquidity.

The traditional multifactor risk model is one of the most commonly used multifactor stock selection strategies. It follows three stages to obtain quantitative strategies: factor screening, discovery, and stock selection. With the advancement of machine learning algorithms and artificial intelligence technologies, reforms have been made to stock selection and discovery modelling. Some examples include discovering new factors from data from social media using text mining technologies and machine learning stock selection models. However, the author mentions that these stock selection models have limited practical application due to challenges such as low-signal-to-noise ratio (SNR), non-independent identical distribution, and the nature of time series. The author also mentions that stock return prediction is considered a regression issue for the neural network algorithm, with time series data and deep learning algorithms mainly used [36].

According to the author, SNR is a better measure of distortion in the time domain between the reconstructed and original signals. The author notes that financial data in time series format is often accompanied by a large fraction of unusual and nonstationary noise, leading to ”low SNR.” Thus, it is challenging for the regression model to capture suitable reconstructed signals in the training process to obtain a final prediction with minimal errors. Additionally, predicting stock price movements (up or down) can be considered a classification problem. The author suggests that classification algorithms such as convolutional neural networks (CNNs), random forests, and support vector machines can be useful to achieve a high level of accuracy, but they may not always result in high returns in the trading strategy. For instance, the author notes that a correct prediction with the model might yield minimal gain, while an occasional incorrect prediction could result in a huge loss, or significant downside risk may exist.

The paper also analyzes the stock selection task from investors' perspectives and relative views, ranking them to evaluate stock performance. The data for this analysis was collected from the China and USA stock markets.

Table 2.2: LTR in Finance Literature Review Summary

Citation	Features	Model	Dataset
[29]	1) sentiment shock score, 2) sentiment trend score, 3)1-week leading return, 4) 1-month leading return, 5) 1-week leading average sentiment, and 6) 1-month leading average sentiment.	ListNet, RankNet	Bloomberg terminal and Thomson Reuters News Analytics
[30]	Raw Data	LSTM	Historic price data for China's A-share market
[31]	68 factors (Table 1 in [31])	ListFold (MLP)	Historic price data for China's A-share market
[28]	Price Data and Graph Relations	LSTM + Node2Vec	Stocks from New York Stock Exchange (NYSE) and NASDAQ stock exchange. Relational Graphs from Wikidata.
[32]	Price Data and Technical Indicators	LSTM-RNN and MLP with Attention	Chinese stock market and the American stock market.

[35]	1) Company identification information, 2) equity profile, 3) company overview information, and 4) company capital-changes history, 5) financial ratios, and 6) price data	ML regression models, LTR methods such as Lamb-daMART, LambdaRank, RankBoost, RankNet, and ListNet	Saudi Stocks Exchange ”Tadawul”
[36]	Price Data	LTR methods such as MART, Coordinate Ascent, RankNet, RFRanker, and TS-Deep-LtM	Chinese stock market and the American stock market.

Chapter 3

Dataset and Data Preprocessing

3.1 Data Collection

This study utilizes historical stock data from the S&P 500 index as the primary dataset for analysis. The S&P 500 index, a widely recognized benchmark of the U.S. stock market, comprises 500 leading companies listed on stock exchanges in the United States [37] . S&P 500 stands for the Standard & Poor's 500 and is considered as the best indicator of how the U.S. economy is performing overall.

The dataset was obtained from multiple reliable financial data providers, as suggested by literature on financial data sources. Additionally, the dataset was cross-verified with the official S&P 500 index data from Standard & Poor's.

3.2 Data Characteristics

This section presents an overview of the key characteristics of the S&P 500 dataset, which is essential for understanding the data's structure, features, and quality, as emphasized by [38].

The S&P 500 dataset comprises daily observations of 500 companies listed on major U.S. stock exchanges. Each observation includes the following variables, as shown in 3.1, which are commonly used in financial studies [39]

- Company Ticker Symbol: A unique identifier for each company.
- Date: The trading date.
- Open: The opening stock price.
- High: The highest stock price during the trading day.
- Low: The lowest stock price during the trading day.
- Close: The closing stock price.
- Adjusted Close: The closing stock price adjusted for dividends and stock splits.
- Volume: The number of shares traded during the trading day.

	Date	Open	High	Low	Close	Adj Close	Volume	Ticker
0	1990-02-16	0.000000	0.079861	0.073785	0.077257	0.054863	940636800.0	CSCO
1	1990-02-20	0.000000	0.079861	0.074653	0.079861	0.056712	151862400.0	CSCO
2	1990-02-21	0.000000	0.078993	0.075521	0.078125	0.055479	70531200.0	CSCO
3	1990-02-22	0.000000	0.081597	0.078993	0.078993	0.056095	45216000.0	CSCO
4	1990-02-23	0.000000	0.079861	0.078125	0.078559	0.055787	44697600.0	CSCO
...	...	...	...	...	...	...	...	...
4113904	2022-10-18	25.299999	25.580000	25.250000	25.389999	25.389999	4184800.0	PPL
4113905	2022-10-19	25.110001	25.459999	24.990000	25.330000	25.330000	4962500.0	PPL
4113906	2022-10-20	25.350000	25.440001	24.790001	24.990000	24.990000	4805400.0	PPL
4113907	2022-10-21	25.150000	25.770000	24.950001	25.690001	25.690001	6585100.0	PPL
4113908	2022-10-24	25.879999	26.010000	25.309999	25.459999	25.459999	4930600.0	PPL

4113909 rows × 8 columns

Figure 3.1: Collected Dataset for S&P 500 stocks

As shown in 3.2, the dataset's descriptive statistics offer insights into the central tendency, dispersion, and distribution of the data [38]. Key statistics, such as mean, median, standard deviation, skewness, and kurtosis, have been calculated for each numeric variable to provide a comprehensive understanding of the dataset's characteristics.

Also, the following are other characteristics of the dataset:

- The dataset consists of 500 distinct stock IDs
- Dates starts from 1962-01-02 to 2022-10-24

	Open	High	Low	Close	Adj Close	Volume
count	4.113909e+06	4.113909e+06	4.113909e+06	4.113909e+06	4.113909e+06	4.113909e+06
mean	4.868898e+01	4.959736e+01	4.842432e+01	4.902451e+01	4.170736e+01	5.257364e+06
std	1.165941e+02	1.179304e+02	1.149886e+02	1.164816e+02	1.128242e+02	2.884545e+07
min	0.000000e+00	1.302100e-02	1.237000e-02	1.302100e-02	1.858731e-03	0.000000e+00
25%	7.833333e+00	8.450000e+00	8.218750e+00	8.333333e+00	4.101000e+00	4.399000e+05
50%	2.431000e+01	2.472500e+01	2.409500e+01	2.441919e+01	1.637696e+01	1.338000e+06
75%	5.324500e+01	5.387000e+01	5.264000e+01	5.326000e+01	4.274483e+01	3.586500e+06
max	5.977610e+03	5.982450e+03	5.884060e+03	5.959330e+03	5.959330e+03	7.421641e+09

Figure 3.2: Collected Dataset Description

3.3 Data Preprocessing

The quality of the S&P 500 dataset is crucial for ensuring the reliability and validity of the study's findings [38]. The data quality has been assessed in terms of accuracy, consistency, completeness, and timeliness. Data accuracy and consistency were ensured by obtaining the dataset from reputable financial data providers and cross-verifying the data with official sources. Data completeness was addressed during the preprocessing stage by identifying and handling missing values [40]. The dataset's timeliness was achieved by using the most recent and up-to-date data available at the time of the study.

The data preprocessing stage is crucial for preparing the raw S&P 500 dataset for the learning to rank algorithms and GBMs. This stage involves cleaning, feature engineering, normalization, and partitioning the dataset, as recommended by [38] and [40].

During the preprocessing stage, it was observed that stocks have different start dates for historical data, given that they were listed on the stock exchange at different times over the

years. Some stocks in the dataset have historical data going back to 1962, while others have less than 5 years of data. This observation is consistent with the findings of Reilly and Brown [37], who noted that the composition of stock indices changes over time due to various factors, such as mergers, acquisitions, and delistings.

A study was conducted to visualize the timeline of stocks entering the dataset at different times, shown in Figure 3.3, as suggested by Kirkpatrick & Dahlquist [41]. The goal of this study was to select stocks with fixed-length historical data and no missing values to avoid introducing biases by imputing these missing values, in line with the recommendations of Hair [38].

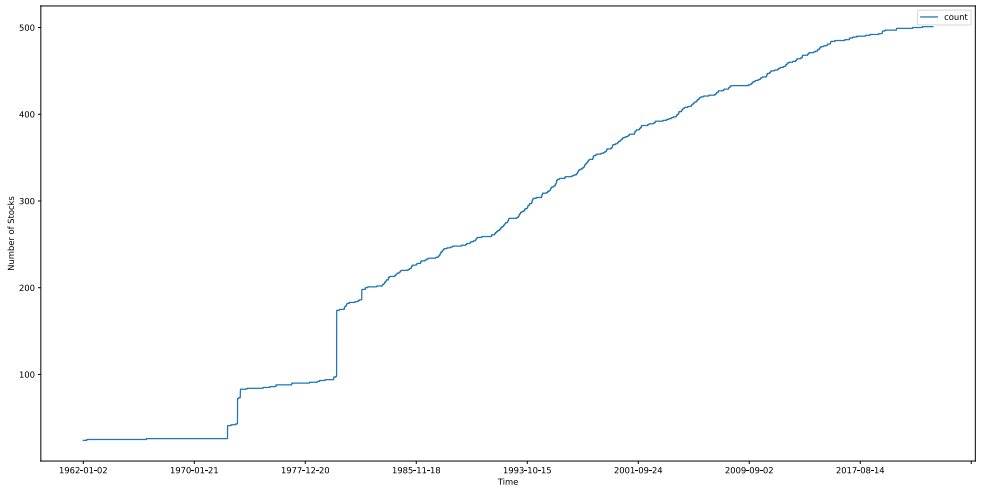


Figure 3.3: Stock Existence Timeline

After analyzing the trade-off between the number of stocks to be included and the length of historical data needed for the model to learn patterns, it was decided to use a fixed-length dataset spanning two decades [40]. This decision resulted in a final dataset consisting of 354 stocks with complete historical data going back to before the year 2000.

In conclusion, the data preprocessing stage successfully transformed the raw S&P 500 dataset into a clean dataset with 354 stocks and no missing values, providing a solid foundation for the implementation and evaluation of the learning to rank algorithms and GBM in stock ranking analysis.

3.4 Feature Generation and Selection

3.4.1 Importance of Feature Generation

Feature generation, also known as feature engineering, is a critical step in the machine learning process that significantly impacts the performance of predictive models. It involves the creation of new features or the transformation of existing features to enhance the predictive power of the dataset. The main objective of feature generation is to capture valuable information from the raw data and provide the learning algorithms with meaningful input features that improve their generalization capabilities. A well-crafted set of features can lead to better model performance, faster training times, and increased interpretability, which is essential for understanding the underlying relationships between the input variables and the target variable [42].

3.4.2 Feature Selection

On generating features, it was taken into consideration having features that capture Momentum, Trend, Statistical Properties, Volume, Overlapping , Volatility, Performance, and Datetime features. A more detailed demonstration example is as follows:

- Momentum (RSI, MACD, ROC, UO, Willr)
- Trend (ADX)
- Statistics (Kurtosis, Skew, STD, Variance, Z Score)
- Volume (OBV, MFI)
- Overlapping (SMA, MA, EMA)
- Volatility (ATR, volatility)
- Performance (Drawdown, Log_Return)

- Datetime (Date day, Date weekday, Date week, Date quarter, Date month, and Date year)

The following are the features selected and generated:

- RSI (Relative Strength Index) measures the strength of a security's price action relative to its past performance.
- 26_ema (26-day Exponential Moving Average) is the average price of a security over the past 26 days, with more weight given to recent prices.
- 12_ema (12-day Exponential Moving Average) is the average price of a security over the past 12 days, with more weight given to recent prices.
- MACD (Moving Average Convergence Divergence) is a trend-following momentum indicator that calculates the difference between two Exponential Moving Averages.
- Signal is a line that is plotted alongside the MACD to indicate when to buy or sell a security.
- Hist (Histogram) represents the difference between the MACD and Signal line.
- Momentum_14 is the difference between the current closing price and the closing price 14 days ago.
- ROC_14 (Rate of Change) is the percentage change in price over the past 14 days.
- UO (Ultimate Oscillator) combines three time frames of Relative Strength Index to smooth out fluctuations and provide a more accurate picture of trend strength.
- Willr is the Williams %R oscillator that measures whether a security is overbought or oversold.
- SMA_14 (Simple Moving Average) is the average price of a security over the past 14 days, with equal weight given to all prices.
- High_low_average is the average of the high and low prices of a security.

- MA5, MA10, MA20, MA50, MA100, MA200 (Moving Averages) are different time frames of Simple Moving Averages.
- EMA5, EMA10, EMA20, EMA50, EMA100, EMA200 (Exponential Moving Averages) are different time frames of Exponential Moving Averages.
- Rolling_std (Rolling Standard Deviation) measures the volatility of a security over a given period.
- Upper_band and Lower_band are lines that are plotted two standard deviations away from the Simple Moving Average to indicate the upper and lower limits of normal price fluctuations.
- Volatility is a measure of the variation in price over a given period.
- ATR (Average True Range) measures the volatility of a security based on its high, low, and close prices.
- ADX_14 (Average Directional Index) measures the strength of a security's trend.
- Kurtosis measures the peakedness of a security's price distribution.
- Skew_30 measures the asymmetry of a security's price distribution over the past 30 days.
- STD (Standard Deviation) measures the deviation of a security's prices from the mean.
- Variance measures the dispersion of a security's prices around the mean.
- Zscore measures the number of standard deviations a security's price is from the mean.
- OBV (On-Balance Volume) measures buying and selling pressure based on volume.
- MFI (Money Flow Index) measures buying and selling pressure based on price and volume.
- Drawdown measures the peak-to-trough decline of a security's value.

- Log_return_14 measures the natural logarithm of the percentage change in price over the past 14 days.
- Date_day, Date_weekday, Date_week, Date_quarter, Date_month, and Date_year are features related to the date of the stock price data.

3.4.3 Datetime Features

Incorporating datetime features, such as day, weekday, week, quarter, month, and year, is crucial for capturing the temporal patterns and dependencies in time-series data. These features enable the model to account for the impact of time-related factors on stock prices, such as seasonality, market cycles, and structural changes. Including datetime features can also help the model identify and exploit recurring patterns and trends that may be associated with specific periods or events, such as earnings announcements, dividends, and macroeconomic releases.

3.4.4 Moving Averages

In this study, various moving averages, including simple moving averages (SMAs) and exponential moving averages (EMAs), with periods ranging from 5 to 200 days, were added as features to the dataset. The rationale behind incorporating multiple moving averages is to leverage their ability to capture temporal patterns, trends, and seasonality in the stock prices, similar to the strengths of Long Short-Term Memory (LSTM) networks and Recurrent Neural Networks (RNNs) [43]. By using these moving averages, the GBM can exploit the advantages of both sequential data models and its own inherent strengths, such as handling non-linear relationships, variable interactions, and robustness to outliers[5] (Chen Guestrin, 2016).

In conclusion, feature generation and selection are essential components of the machine learning process that can significantly enhance the performance of predictive models. By incorporating datetime features and various moving averages, this study aims to create a comprehensive and informative dataset that enables the GBM to capture the complex temporal patterns and dependencies inherent in stock price data.

3.5 Final Dataset

During the data preprocessing phase, several steps were taken to clean and enhance the dataset. These steps included handling missing values, correcting data inconsistencies, and transforming variables as needed. Also, this involves the careful selection of a suitable timeline to strike a balance between the number of stocks in the dataset and the depth of the historical data. Additionally, a variety of features were generated using calculations and derivations from the original dataset. This process of feature engineering aimed to provide valuable information that could potentially help improve the predictive performance of the machine learning models.

The final dataset, illustrated in Figure 3.4 , is a result of this meticulous preprocessing and feature generation process. It is a comprehensive and well-structured data source that can be used for model training, evaluation, and further analysis. By carefully selecting the data range and creating meaningful features, this dataset serves as a strong foundation for the subsequent machine learning tasks and evaluations. Figure 3.5 shows a statistical description of the numerical features in the final dataset.

	Date	Open	High	Low	Close	Adj Close	Volume	Ticker	rsi	26_ema	...	obv	mfi	d
0	2000-01-03	0.936384	1.004464	0.907924	0.999442	0.853356	535796800.0	AAPL	64.526573	0.764035	...	7.094238e+10	57.349628	
1	2000-01-04	0.966518	0.987723	0.903460	0.915179	0.781409	512377600.0	AAPL	52.469860	0.765322	...	7.043000e+10	55.983204	
2	2000-01-05	0.926339	0.987165	0.919643	0.928571	0.792843	778321600.0	AAPL	53.942764	0.767361	...	7.120832e+10	69.554214	
3	2000-01-06	0.947545	0.955357	0.848214	0.848214	0.724232	767972800.0	AAPL	44.942894	0.764166	...	7.044035e+10	57.289150	
4	2000-01-07	0.861607	0.901786	0.852679	0.888393	0.758538	460734400.0	AAPL	49.481434	0.763749	...	7.090109e+10	48.521786	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2031955	2022-10-18	52.959999	53.389999	51.520000	51.980000	51.980000	1541600.0	ZION	46.866283	52.734356	...	2.718983e+08	51.841366	3
2031956	2022-10-19	51.320000	51.610001	48.680000	49.220001	49.220001	2049500.0	ZION	38.723255	52.474033	...	2.698488e+08	51.033485	3
2031957	2022-10-20	49.180000	49.639999	46.680000	47.130001	47.130001	2301200.0	ZION	33.917408	52.078178	...	2.675476e+08	49.744896	4
2031958	2022-10-21	47.180000	48.500000	46.580002	47.820000	47.820000	2381100.0	ZION	36.710075	51.762756	...	2.699287e+08	40.043003	4
2031959	2022-10-24	48.320000	50.049999	48.130001	49.779999	49.779999	2712400.0	ZION	43.955410	51.615887	...	2.726411e+08	41.873375	3

2031960 rows × 53 columns

Figure 3.4: Final Dataset

	Open	High	Low	Close	Adj Close	Volume	rsi	26_ema	12_ema	mac
count	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+06	2.031960e+0
mean	6.718571e+01	6.795471e+01	6.639577e+01	6.719492e+01	5.831482e+01	6.781810e+06	5.259272e+01	5.802700e+01	5.818957e+01	1.625542e-0
std	1.370251e+02	1.387674e+02	1.352778e+02	1.370549e+02	1.326470e+02	3.356650e+07	1.177971e+01	1.318551e+02	1.322925e+02	2.827796e+0
min	6.041700e-02	6.104200e-02	5.395800e-02	6.104200e-02	6.104200e-02	0.000000e+00	6.688068e+00	6.467561e-02	6.347248e-02	-4.220713e+0
25%	2.293000e+01	2.324000e+01	2.260417e+01	2.293413e+01	1.635000e+01	9.417000e+05	4.446496e+01	1.632384e+01	1.634227e+01	-2.011083e-0
50%	4.112000e+01	4.161500e+01	4.063000e+01	4.114000e+01	3.133509e+01	2.118800e+06	5.286590e+01	3.124085e+01	3.129315e+01	9.517983e-0
75%	7.215000e+01	7.291000e+01	7.138000e+01	7.217000e+01	6.092893e+01	5.056800e+06	6.093720e+01	6.063101e+01	6.078278e+01	4.715066e-0
max	5.977610e+03	5.982450e+03	5.884060e+03	5.959330e+03	5.959330e+03	7.421641e+09	9.561759e+01	5.700242e+03	5.823229e+03	2.066993e+0

8 rows × 51 columns

Figure 3.5: Final Dataset Description

3.6 Model's Input vs Output

The input to the model consists of a set of features derived from historical stock data for S&P 500 stocks. These features are selected to be diverse and to capture different patterns, relationships, and sentiment of the financial market as well as stocks' behaviors. By incorporating a diverse set of features, the model aims to capture various patterns and trends in stock prices.

The ground truth for the model is based on the actual future stock prices. In this study, the future price is calculated by shifting the adjusted closing price of the stock by a predetermined number of days (a month, a quarter, and a year). See Equation (3.1) for the calculation of return. P_{t-1} is the `adj close` for day t. By comparing the model's predictions to these real-world future prices, the model can be evaluated and fine-tuned to improve its accuracy.

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \tag{3.1}$$

The expected output of the model is a ranking of stocks based on their predicted future returns. The GBM model is trained to predict the percentage return of each stock over a specified period (21 days). Using these predictions, the stocks can be ranked, allowing investors to identify the most promising stocks in terms of potential returns.

3.7 Summary

This chapter covers the dataset and data preprocessing of the stock ranking model, starting with data collection from the S&P 500 index. The dataset characteristics are discussed, including key variables and descriptive statistics. Data preprocessing involves cleaning, feature engineering, normalization, and partitioning to ensure data quality. Features are generated and selected to capture various aspects of stock price movements, including momentum, trend, statistical properties, volume, and datetime information. The input to the model consists of these diverse features, while the ground truth is stocks' returns one month ahead, 21 weekdays which is equivalent to one financial month. The model's output is a ranking of stocks based on their predicted future returns. Moving averages and datetime features help capture temporal patterns and dependencies in the data. Overall, the chapter provides a comprehensive overview of the data preparation process, setting the foundation for building and evaluating the stock ranking model using learning to rank algorithms and GBMs.

Chapter 4

Proposed Model

4.1 Gradient Boosting Machine

4.1.1 What is GBM?

GBM is an ensemble learning method that combines multiple weak learners, typically decision trees, to form a strong learner. GBM is a robust machine learning algorithm that is commonly used in a wide range of applications, including classification, regression, and ranking problems. GBMs are based on the idea of iteratively improving the performance of an ensemble of weak learners by combining their predictions in a weighted manner. The basic idea is to fit a sequence of models, each one correcting the errors of the previous one based on the loss function. At each iteration, a new model is trained to predict the negative gradient of the loss function concerning the output of the previous model, so that the overall output is moved closer to the true values [44].

4.1.2 Architecture of a GBM

GBMs consist of six main components: objective function, weak learners, gradient descent, ensemble learning, regularization, and hyperparameters. They are briefly explained as follows:

1. Objective function: its primary goal is to minimize the loss function by continuously adding new weak learners to the ensemble, usually using the mean squared error as the loss function.

2. Weak learners: refer to models that slightly outperform random guessing, with decision trees being the most commonly used in GBMs.

3. Gradient descent: used to improve the model iteratively by minimizing the gradient of the loss function with respect to the predicted values

4. Ensemble learning: utilized to combine the predictions of multiple weak learners to make the final prediction

5. Regularization techniques: such as shrinkage or tree pruning are used to prevent overfitting by reducing the contribution of each new weak learner or removing unhelpful nodes

6. Hyperparameters: including the number of weak learners, learning rate, maximum tree depth, minimum node split samples, and regularization parameters, and they require tuning to achieve optimal performance.

4.1.3 How does it operate?

The GBM algorithm involves several steps to build a model. Firstly, the model is initialized with a constant value, which is the mean of the target variable of the training data. This value is used as the initial prediction for all instances in the dataset. Next, the GBM algorithm calculates the gradient of the loss function with respect to the current predictions. In the case of squared error loss function, the gradient is calculated as the difference between the current prediction and the actual target value. The GBM algorithm then builds a decision tree to model the relationship between the features and the gradient. Each tree is built by recursively splitting the data into subsets based on the feature values, using the mean squared error reduction criterion. To control the rate of learning and the complexity of the trees, the GBM algorithm uses a learning rate and

tree constraints. The learning rate is a scaling factor applied to the predictions of each tree, which slows down the learning process and improves the stability of the model. The tree constraints include the maximum depth of the tree, the minimum number of samples required to split a node, and the minimum number of samples required to be at a leaf node.

After building a tree, the GBM algorithm calculates the prediction of the tree for each instance in the dataset. The prediction is then multiplied by the learning rate and added to the current prediction. This process of building trees and updating the predictions is repeated for a specified number of iterations, or until the loss function has converged. Once the algorithm has converged, the final predictions are obtained by summing the predictions from all the trees.

To prevent overfitting, the GBM algorithm uses regularization techniques such as tree pruning and shrinkage. Tree pruning involves removing nodes from the decision tree that do not improve the model's performance, while shrinkage reduces the contribution of each new tree to the overall prediction. In the case of GBM, the shrinkage is controlled by the learning rate, which is set to a small value to reduce the impact of each tree.

Finally, the GBM algorithm has several hyperparameters that need to be tuned to optimize its performance. Commonly used hyperparameters include the learning rate, maximum depth of the trees, and the number of trees. In practice, cross-validation is often used to tune these hyperparameters.

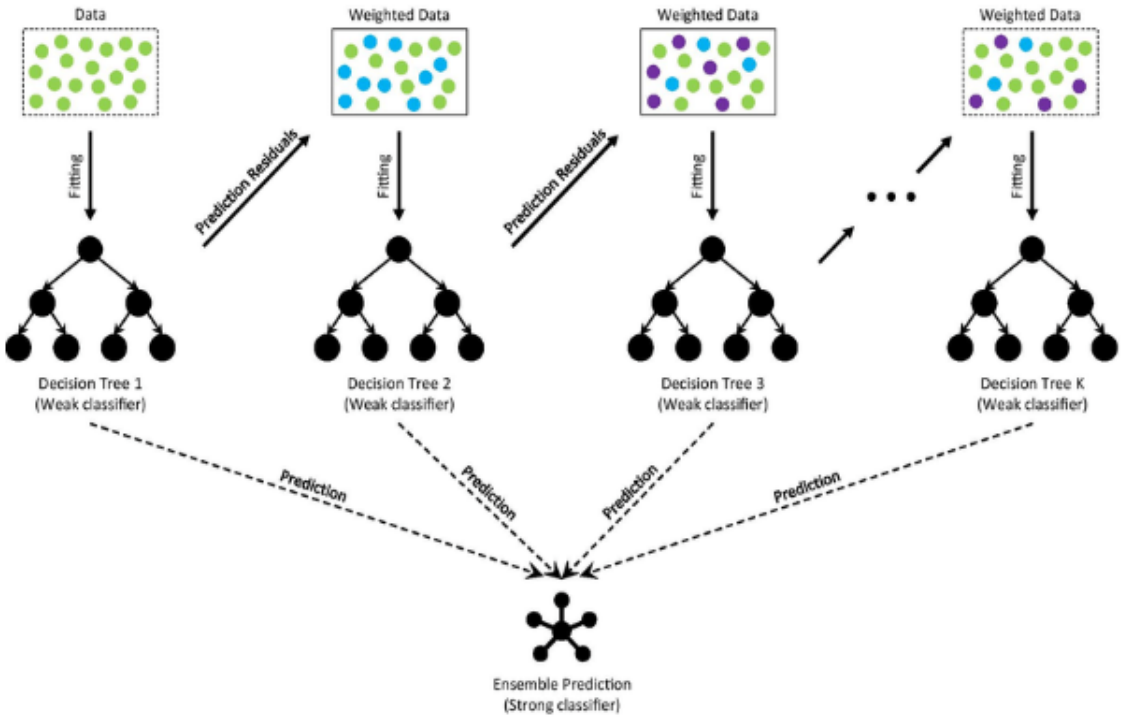


Figure 4.1: How Gradient Boosting Decision Tree operates

4.1.4 Why GBM?

Using GBM, specifically, Catboost is suitable for the problem. CatBoost is effective in handling categorical features and imbalanced data, which are common in financial datasets [45]. Figure 4.2 in [45] and rest of paper shows that CatBoost outperforms other gradient boosting frameworks such as XGBoost and LightGBM on several benchmarks, including the Microsoft Learning to Rank dataset and the Higgs Boson dataset. CatBoost can handle missing values without imputations, which is an advantage in financial data where missing data is common. Moreover, CatBoost have the ability to improve prediction performance even when the dataset includes high-dimensional features, which is also relevant in the context of financial data. Therefore, CatBoost is a suitable choice for developing stock portfolio selection strategies in the financial domain.

	CatBoost	LightGBM	XGBoost
Adult	**0.270 / 0.127**	+2.4% / +1.9%	+2.2% / +1.0%
Amazon	**0.139 / 0.044**	+17% / +21%	+17% / +21%
Click	**0.392 / 0.156**	+1.2% / +1.2%	+1.2% / +1.2%
Epsilon	**0.265 / 0.109**	+1.5% / +4.1%	+11% / +12%
Appetency	**0.072 / 0.018**	+0.4% / +0.2%	+0.4% / +0.7%
Churn	**0.232 / 0.072**	+0.1% / +0.6%	+0.5% / +1.6%
Internet	**0.209 / 0.094**	+6.8% / +8.6%	+7.9% / +8.0%
Upselling	**0.166 / 0.049**	+0.3% / +0.1%	+0.04% / +0.3%
Kick	**0.286 / 0.095**	+3.5% / +4.4%	+3.2% / +4.1%

Figure 4.2: Comparison with baselines: log loss / zero-one loss (relative increase for baselines).

4.1.5 GBM strengths

GBMs have several advantages and strong abilities when compared with other candidates given the task of stocks ranking. Other candidates in the field ranges from ML to DL tools. The strengths of GBMs are as follows:

- **High Predictive Power:** GBMs are known for their high predictive power and are often used to solve complex problems with high-dimensional data. GBMs are capable of achieving high accuracy on a wide range of tasks, including classification, regression, and ranking [46].
- **Feature Importance:** GBMs provide a feature importance ranking that can be used to identify the most important variables in the data. This ranking can be used to remove irrelevant variables or to improve the accuracy of other models. the feature importance ranking provided by GBMs is particularly useful in the context of high-dimensional data, where it can be difficult to identify the most important variables [46].
- **Robustness to Outliers:** GBMs can still produce accurate results even when the data contains extreme values.GBMs can effectively handle outliers by adjusting the model to focus on the most informative data points.

- **Less Prone to Overfitting:** GBMs have a regularization parameter (e.g., shrinkage, max depth, min child weight) that can help prevent overfitting.
- **Fast prediction speed:** GBMs can make predictions very quickly once they have been trained, which can be an advantage in real-time applications.
- **Interpretability:** GBMs provide interpretable feature importance measures, making it easier to understand the model's predictions and identify the most important factors affecting the outcome.

Providing GBMs with additional features, such as moving averages going back up to 200 days can certainly help capture temporal dependencies in the data. Additionally, providing GBMs with features that capture stocks' momentum, trend, and seasonality will improve the model's performance. The result is providing a very powerful machine learning model that combines advantages of its own with other tools that are known for their strength in capturing temporal dependencies of a sequential data.

4.2 Learning to Rank

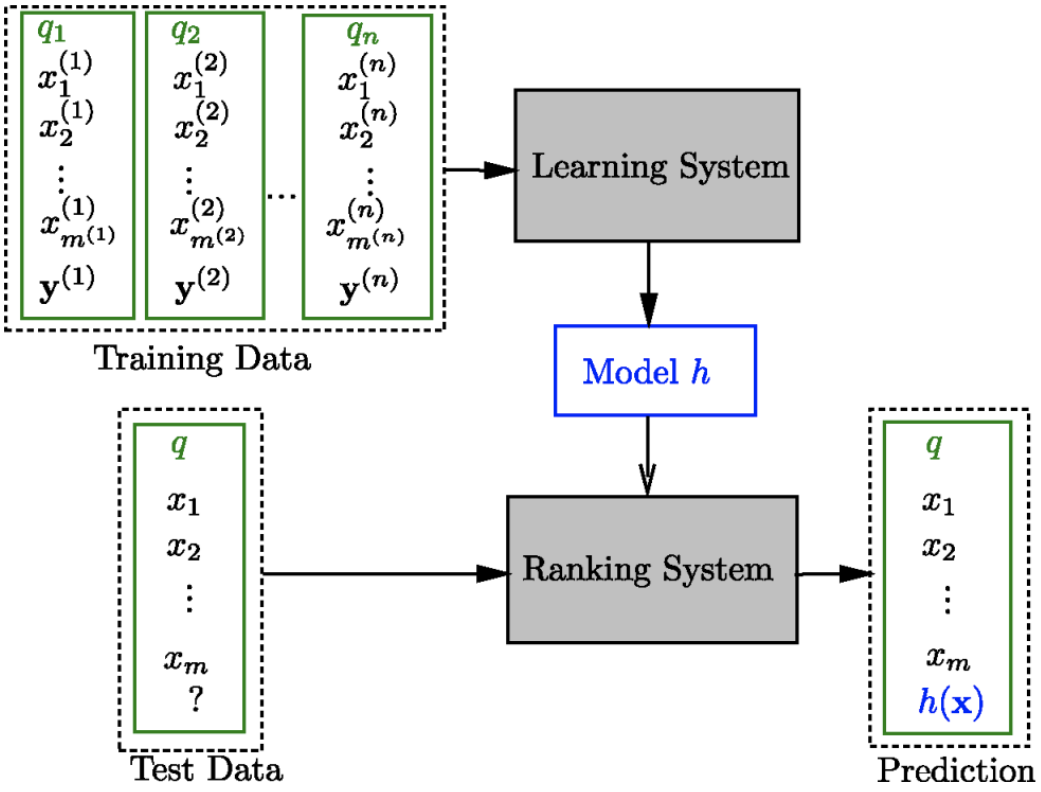


Figure 4.3: LTR framework

LTR is a subfield of machine learning that focuses on developing algorithms to rank items or objects in a specific order. The primary goal of LTR is to learn a model that can produce an optimal ranking of items based on their relevance, importance, or other criteria. LTR has gained significant attention in recent years due to its applicability in various real-world problems, such as information retrieval, recommendation systems, and financial market prediction.

Stock ranking algorithm aims to demonstrate the relative level of investment revenues of stocks by giving a ranking prediction based on the various stock features. The general definition can be described as:

$$\widehat{s} = f(x) \tag{4.1}$$

where x is the input features of stocks and $\widehat{s}$ is the predicted ranking values.

Given the data and task in hand, learning to rank algorithms can be categorized into Point-wise, Pairwise, and Listwise. There is a trade-off in these three approaches between complexity of model and performance, as shown in Figure 4.4.

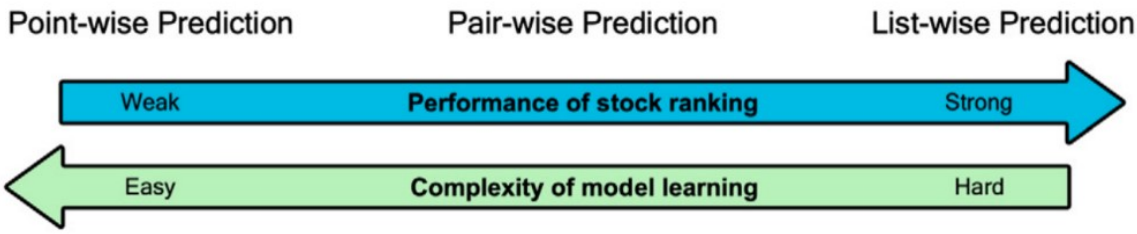


Figure 4.4: Complexity-Performance Trade off between three methods of LTR

As present in Figure 4.5, they also differ in how they handle the input, training, and outputting the prediction.

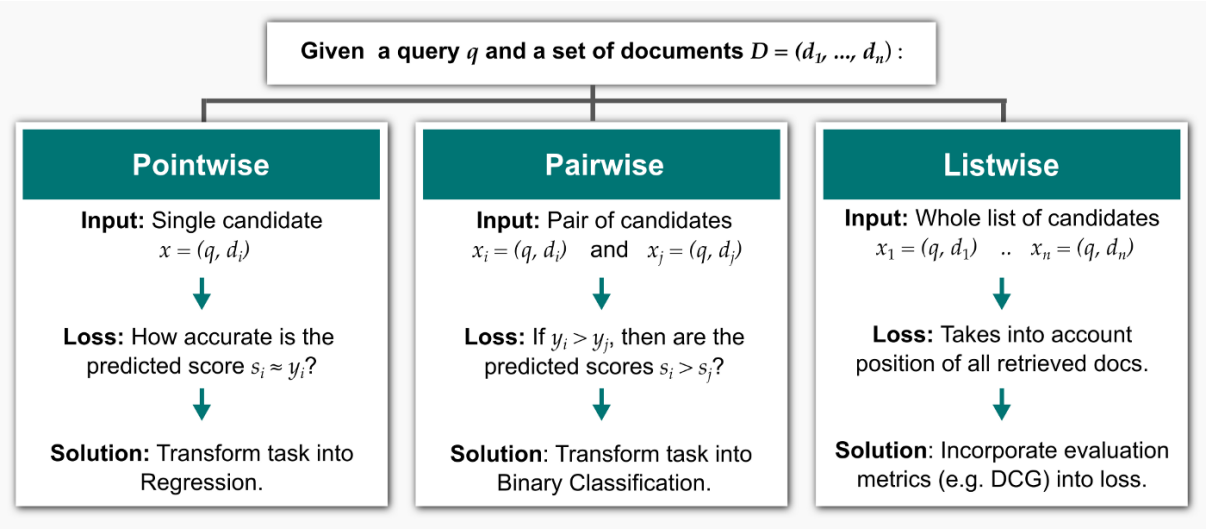


Figure 4.5: Comparison between three methods of LTR

4.2.1 Pointwise approach

The pointwise approach treats the learning to rank problem as a regression or classification task [47]. It assigns a relevance score or label to each document individually, without considering the context of other documents in the query. The goal is to predict the relevance score for each

document-query pair and then rank the documents based on these scores. Common pointwise learning algorithms include linear regression, support vector machines (SVM), and neural networks.

Input: The input to the pointwise approach consists of feature vectors that represent the documents, query, and relevance score. Each instance is a triple consisting of a query, document, and relevance score.

Training: In a pointwise approach, the training process optimizes the model to predict the relevance score for each document-query pair. The training process involves learning a function that maps the features of each instance to its corresponding relevance score.

Output: The output of the pointwise approach is a list of relevance scores for each document-query pair. The documents are then ranked based on these scores.

Assuming that there are K stocks in total, then the procedures of pointwise LTR prediction can be described as:

- Point-wise: $\widehat{s}_k = f_o^k(x_k)$

  where $k = 1, 2, 3, ..., K$ and $f_o^k(x_k)$ is the point-wise prediction function

4.2.2 Pairwise approach

The pairwise approach focuses on the relative order of document pairs rather than individual document scores. The objective is to minimize the number of incorrectly ordered document pairs. In this method, a preference is learned for each pair of documents, and a ranking is derived based on these preferences. Popular pairwise learning algorithms include RankNet [7] and RankSVM [48].

Input: The input to the pairwise approach consists of pairs of documents and the query. Each instance is a triple consisting of a query, a pair of documents, and the pairwise preference between them.

Training: In the pairwise approach, the training process involves learning a function that predicts the relative order of pairs of documents. The training process involves minimizing the

number of incorrectly ordered document pairs.

Output: The output of the pairwise approach is a ranked list of documents based on the learned preferences for each pair of documents.

Assuming that there are K stocks in total, then the procedures of pairwise LTR prediction can be described as:

- Pair-wise: $(\widehat{s}_i, \widehat{s}_j) = f_p(x_i, x_j)$

  where $i, j = 1, 2, 3, ..., K$ and $f_p(x_i, x_j)$ is the pair-wise prediction function

4.2.3 Listwise approach

The listwise approach considers the entire list of documents as a whole and learns to optimize the ranking of the entire list. The goal is to directly optimize a ranking function that directly maps the query and the list of documents to their corresponding relevance scores. Common listwise learning algorithms include ListNet [49] and ListMLE [50].

Input: The input to the listwise approach consists of a query and a list of documents. Each instance is a pair consisting of a query and a list of documents.

Training: In the listwise approach, the training process optimizes a function that directly maps the query and the list of documents to their corresponding relevance scores. The training process involves learning a function that can rank the entire list of documents directly.

Output: The output of the listwise approach is a ranked list of documents based on the learned ranking function. The ranking function directly maps the query and the list of documents to their corresponding relevance scores.

Assuming that there are K stocks in total, then the procedures of listwise LTR prediction can be described as:

- List-wise: $\widehat{s} = [\widehat{s}_1, \widehat{s}_2, ..., \widehat{s}_k] = f_l[x_1, x_1, ..., x_k]$

  where $k = 1, 2, 3, ..., K$ and $f_l[x_1, x_1, ..., x_k]$ is the list-wise prediction function

4.2.4 Summary

In summary, LTR algorithms consisting of its three main approaches—pointwise, pairwise, and listwise—each with its own strengths and weaknesses when applied to stock ranking. The pointwise approach treats the problem as a regression or classification task, predicting relevance scores for individual stocks without considering the relationships between them. Although it is simpler to implement, the pointwise method may not be the best choice for stock ranking, as it does not explicitly focus on the ordering of stocks. On the other hand, the pairwise approach emphasizes the relative order of stock pairs, minimizing the number of incorrectly ordered pairs, while the listwise approach directly optimizes a ranking function considering the entire list of stocks. Both pairwise and listwise methods are generally more suitable for stock ranking, as they capture the relationships between stocks and directly optimize ranking-based loss functions, providing more accurate and reliable stock rankings.

Chapter 5

Results

This chapter would present the performance comparison of various ranking algorithms, including:

- Regress-then-Rank using RMSE, and learning to rank approaches such as
- `PairLogitPairwise`
- `QueryRMSE (Listwise)`
- `YetiRank`
- `YetiRankPairwise`

The results were obtained by evaluating these methods on a dataset, which was prepared as described in the previous chapters. The performance of each approach based on various evaluation metrics would be discussed and then would compare them to determine the most effective ranking method for our specific application.

5.1 Regress-then-Rank (RMSE)

Regress-then-Rank is a two-step approach where a regression model is first trained to predict the relevance scores, and then the predicted scores are used to rank the items [51]. We used

the RMSE (Root Mean Squared Error) as the evaluation metric for the regression model. The performance of this approach depends on the accuracy of the regression model, which can be influenced by various factors, such as the choice of model, feature engineering, and hyperparameter tuning.

5.2 Learning-to-Rank Pairwise

PairLogitPairwise is a pair-based learning to rank method that optimizes the ranking of pairs of items within query groups [52]. It utilizes logistic loss functions to measure the difference between the predicted order of item pairs and their true order. The main difference between PairLogit and PairLogitPairwise lies in the weighting scheme, with PairLogitPairwise taking into account the number of inversions required to achieve the correct ranking. Specifically, the PairLogitPairwise loss function is defined as:

$$L_{PairLogitPairwise} = \sum_{i,j \in \mathcal{S}} \ln(1 + \exp(-(s_i - s_j))w_{i,j}) + \lambda \sum_{i=1}^{d} \|w_i\|^2 \tag{5.1}$$

where s_i and s_j are the scores of items i and j, respectively, and $w_{i,j}$ is the weight assigned to the pair (i, j). The weight $w_{i,j}$ is calculated as $\frac{1}{n_{i,j}}$, where $n_{i,j}$ is the number of inversions required to rank item i above item j. The regularization parameter λ controls the trade-off between fitting the data and preventing overfitting.

PairLogitPairwise is significantly faster in training and less computationally expensive than PairLogit [52]. However, it may not perform as well as other learning to rank methods on datasets with a large number of queries and/or items due to its reliance on pair-wise comparisons. Nevertheless, it is a useful method to consider when dealing with smaller datasets or when computational resources are limited.

5.3 Learning-to-Rank Listwise

QueryRMSE and YetiRank are listwise loss functions for learning to rank tasks. On the other hand, YetiRankPairwise is based on a combination of listwise and pairwise approaches but ultimately follows the listwise ranking philosophy. They each have different optimization objectives and are based on different ranking approaches.

5.3.1 QueryRMSE

QueryRMSE stands for Query Root Mean Squared Error. It is inspired by the listwise approach to ranking and aims to minimize the root mean squared error between the predicted relevance scores and the true relevance scores within a query group. QueryRMSE focuses on minimizing the difference between the actual and predicted scores while considering the entire list at once. This loss function is more sensitive to larger prediction errors and tends to work well when there is a significant difference between the relevance scores of items within the same query group.

The equation for QueryRMSE is as follows:

$$\text{QueryRMSE} = \frac{1}{N}\sqrt{\sum_{q}\sum_{d_{qk}}\left(f(d_{qk}) - l_{qk} - c_q\right)^2} \tag{5.2}$$

where N is the total number of items in all query groups, $f(d_{qk})$ is the predicted relevance score of item k in query group q, l_{qk} is the true relevance score of item k in query group q, and c_q is the query-level bias term.

5.3.2 YetiRank

In YetiRank, the goal is to order items within each query group based on their predicted relevance scores, with the ultimate objective of maximizing the number of correctly ordered pairs of items. The approach is based on the concept of Gini impurity, which measures the impurity of a dataset by considering how often a randomly chosen item would be incorrectly labeled if

it was labeled based on the distribution of the labels in the dataset.

YetiRank utilizes a gradient-based optimization method, where the objective is to minimize the Gini impurity criterion with respect to the relevance scores of the items. This is achieved by minimizing the negative expected value of the Gini impurity criterion with respect to a probabilistic model of the relevance scores. The optimization is performed using a stochastic gradient descent algorithm, where the gradient is estimated using a small subset of the data at each iteration.

The performance of YetiRank is largely dependent on the quality of the relevance scores, as it aims to maximize the number of correctly ordered pairs of items within each query group. It can work well for large-scale ranking tasks, especially when there is a significant difference between the relevance scores of items within the same query group. However, it may not be the best choice for tasks where the relevance scores are noisy or there is a large number of query groups with few items.

5.3.3 YetiRankPairwise - Hybrid Approach

YetiRankPairwise is a hybrid approach that combines both listwise and pairwise methods. Like YetiRank, it also focuses on optimizing the ranking of items within each query group while considering the entire list at once. However, the optimization in YetiRankPairwise is performed by maximizing the likelihood of the correct permutation within each query group.

To achieve this, YetiRankPairwise uses the logistic loss function to measure the difference between the predicted order of item pairs and their true order. The difference between YetiRank and YetiRankPairwise lies in their weighting schemes. YetiRankPairwise takes into account the number of inversions required to achieve the correct ranking, which allows it to handle pairwise interactions between items in a more nuanced way.

YetiRankPairwise also has the advantage of being robust to noise in the relevance scores, making it suitable for large-scale ranking tasks. However, it may be slower to train than simpler pairwise ranking methods, as it requires the computation of the full permutation probabilities

for each query group.

5.4 Performance Comparison

5.4.1 Metrics

5.4.1.1 NDCG

NDCG is a measure of the effectiveness of a ranking system that takes into account the position of relevant items in the ranked list. The equation for NDCG is as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{5.3}$$

where:

- k is the position in the ranked list being evaluated
- $DCG@k$ is the Discounted Cumulative Gain at position k, which is defined as:

$$DCG@k = rel_1 + \sum_{i=2}^{k} \left[\frac{2^{rel_i} - 1}{\log_2(i)} \right]$$

  where rel_i is the relevance score of the ith item in the ranked list. The first item in the list is at position 1, the second item is at position 2, and so on.

- $IDCG@k$ is the Ideal Discounted Cumulative Gain at position k, which is the maximum possible $DCG@k$ for a given set of relevant items. $IDCG@k$ is calculated by sorting the relevant items by their relevance score and then computing the $DCG@k$ for this sorted list.

NDCG is commonly used in information retrieval and recommendation systems to evaluate the effectiveness of different ranking algorithms. It provides a way to compare different ranking

algorithms that produce lists of different lengths. The value of NDCG ranges between 0 and 1, with a higher value indicating a better ranking system.

5.4.1.2 MAP

MAP is a measure of the quality of a ranking system. It is commonly used in information retrieval to evaluate the performance of search engines, but can also be used in other fields such as recommendation systems and machine learning. The equation for MAP is as follows:

$$MAP = \frac{1}{|Q|} \sum_{i=1}^{|Q|} AP(i) \tag{5.4}$$

where:

- Q is the set of queries
- $AP(i)$ is the average precision (AP) for query i
- $|Q|$ is the number of queries in the set Q

The average precision (AP) for a query i is defined as follows:

$$AP(i) = \frac{1}{rel_i} \sum_{j=1}^{|R_i|} P(j) \cdot rel_j$$

where:

- R_i is the set of retrieved documents for query i
- $|R_i|$ is the number of retrieved documents for query i
- $P(j)$ is the precision at rank j
- rel_j is an indicator function that is 1 if the jth document is relevant to query i and 0 otherwise
- rel_i is the total number of relevant documents for query i

The precision at rank j is defined as the number of relevant documents among the top j retrieved documents divided by j.

In practice, MAP is often computed using a set of relevant documents for each query. The relevant documents are typically determined by human evaluators, but can also be determined through automated methods such as relevance feedback or machine learning. MAP value ranges between 0 and 1, with a higher value indicating a better ranking system.

5.4.1.3 RBO

RBO is a measure of the similarity between two ranked lists. The equation for RBO is as follows:

$$rbo(p,q) = (1-p)*S(p,q,0.9)+(1-0.9*p)*S(p,q,0.5)+(1-0.9^2*p)*S(p,q,0.1)+... \quad (5.5)$$

where:

- p is the probability of continuing from a rank in one list to the next rank in that same list
- q is the other list being compared to
- $S(p,q,k)$ is the rbo similarity at depth k

The value of p can be set between 0 and 1, depending on how much weight is given to the higher ranks in the list. A higher value of p gives more weight to the higher ranks, whereas a lower value of p gives more weight to the lower ranks.

RBO measure is typically calculated by approximating the infinite sum using a finite number of terms. The number of terms used will depend on the length of the lists being compared and the desired level of precision.

RBO measure is commonly used in information retrieval to evaluate the similarity between search engine rankings and user preferences. It has also been used in other fields, such as recommendation systems and machine learning.

5.4.2 Performance Comparison: Different Investment Horizons

To compare the performance of the different ranking methods, it was evaluated using various evaluation metrics, such as NDCG, MAP, and Rank Biased Overlap (RBO). These metrics capture different aspects of ranking quality, making it easier to identify the strengths and weaknesses of each approach. NDCG is known for not being a good representative of the bottom ranking items, therefore, to capture the candidates to short in the portfolio, MAP is the metric as it better captures them. Tables 5.1, 5.2, and 5.3 presents the performance of different ranking approaches with different investment horizons using various evaluation metrics: NDCG@4, NDCG@8, NDCG@16, NDCG@32, MAP@-10, and RBO. Analysis of performance on the different models and on different investment horizons are observed and analyzed in the following sections.

5.4.2.1 Monthly Ranking

Table 5.1: Monthly Ranking

	NDCG				MAP	RBO
Approach	**@4**	**@8**	**@16**	**@32**	**@-10**	
Regress-then-Rank	0.3235	0.3801	0.4374	0.5032	0.4751	0.5181
PairLogitPairwise	0.4478	0.4856	0.5280	0.5868	0.4870	0.5014
QueryRMSE Listwise	0.5628	0.5956	0.6278	0.6741	0.5419	**0.5291**
YetiRank Listwise	**0.6113**	**0.6329**	**0.6548**	**0.6887**	**0.5563**	0.5177
YetiRankPairwise Hybrid	0.6089	0.6221	0.6377	0.6623	0.5327	0.5038

Based on the monthly results, we can make the following observations and comparisons:

1. Regress-then-Rank performs significantly worse than the other methods almost across all evaluation metrics. This is not surprising, as learning to rank methods are specifically designed for ranking tasks and can capture the complexities and dependencies of rank order better than regression-based approaches.

2. PairLogitPairwise, a pairwise approach, shows better performance than Regress-then-Rank, but it is outperformed by QueryRMSE Listwise, YetiRank Listwise, and YetiRankPairwise Hybrid. This suggests that listwise approaches may generally be better suited for ranking tasks, as they consider the entire list of items in a query group during optimization.

3. Among the listwise approaches, YetiRank Listwise has the highest performance across most evaluation metrics, followed closely by YetiRankPairwise Hybrid. This indicates that optimizing for the correct ordering of items within a query group, the focus of YetiRank and YetiRankPairwise, is an effective strategy for improving ranking performance.

4. QueryRMSE Listwise also performs well, outperforming pairwise approaches and Regress-then-Rank. However, it is outperformed by YetiRank Listwise and YetiRankPairwise Hybrid. This suggests that while minimizing the root mean squared error can lead to good ranking performance, optimizing for the correct ordering within a query group may yield even better results.

5.4.2.2 Quarterly Ranking

Table 5.2: Quarterly Ranking

	NDCG				MAP	RBO
Approach	**@4**	**@8**	**@16**	**@32**	**@-10**	
Regress-then-Rank	0.5016	0.5335	0.5720	0.6220	0.4830	0.5320
PairLogitPairwise	0.5568	0.5805	0.6047	0.6349	0.4881	0.5120
QueryRMSE Listwise	**0.6033**	**0.6174**	**0.6459**	**0.6747**	**0.5145**	**0.5457**
YetiRank Listwise	0.5646	0.5882	0.6181	0.6545	0.5010	0.5352
YetiRankPairwise Hybrid	0.5708	0.5904	0.6140	0.6462	0.5086	0.5327

1. Regress-then-Rank performs significantly worse than the other methods across almost all evaluation metrics. This is not surprising, as learning to rank methods are specifically designed for ranking tasks and can capture the complexities and dependencies of rank order better than regression-based approaches.

2. PairLogitPairwise, a pairwise approach, shows better performance than Regress-then-Rank, but it is outperformed by QueryRMSE Listwise, YetiRank Listwise, and YetiRankPairwise Hybrid. This suggests that listwise approaches may generally be better suited for ranking tasks, as they consider the entire list of items in a query group during optimization.

3. QueryRMSE Listwise outperforms all other methods, achieving the highest performance across most evaluation metrics. This indicates that minimizing the root mean squared error, as done by QueryRMSE Listwise, is an effective strategy for improving ranking performance in a quarterly setting.

4. YetiRank Listwise and YetiRankPairwise Hybrid also perform well, closely following QueryRMSE Listwise in terms of performance. This suggests that optimizing for the correct ordering of items within a query group, the focus of YetiRank and YetiRankPairwise, remains a strong approach for ranking tasks, even in a quarterly context.

5.4.2.3 Yearly Ranking

Table 5.3: Yearly Ranking

	NDCG				MAP	RBO
Approach	**@4**	**@8**	**@16**	**@32**	**@-10**	
Regress-then-Rank	0.8377	0.8213	0.8071	0.7777	0.4935	0.5665
PairLogitPairwise	0.8836	0.8569	0.8359	0.8127	0.5393	0.5686
QueryRMSE Listwise	0.8526	0.8327	0.8127	0.7837	0.4939	0.5674
YetiRank Listwise	0.9083	0.8701	0.8454	0.8076	0.5409	0.5653
YetiRankPairwise Hybrid	**0.9100**	**0.8956**	**0.8530**	**0.8301**	**0.5791**	**0.5820**

1. Regress-then-Rank, although performing better than in the monthly and quarterly settings, still performs worse than the other methods across almost all evaluation metrics. This reaffirms the idea that learning to rank methods are better suited for ranking tasks due to their ability to capture the complexities and dependencies of rank order.

2. PairLogitPairwise, a pairwise approach, shows better performance than Regress-then-Rank and QueryRMSE Listwise but is outperformed by YetiRank Listwise and YetiRankPairwise Hybrid. This suggests that even in the yearly context, listwise approaches may be better suited for ranking tasks, as they consider the entire list of items in a query group during optimization.

3. YetiRank Listwise and YetiRankPairwise Hybrid demonstrate strong performance, with YetiRankPairwise Hybrid achieving the highest performance across most evaluation metrics. This indicates that optimizing for the correct ordering of items within a query group, the focus of YetiRank and YetiRankPairwise, is particularly effective for improving ranking performance in a yearly setting.

4. QueryRMSE Listwise, although performing better than Regress-then-Rank, is outperformed by both YetiRank Listwise and YetiRankPairwise Hybrid. This suggests that while minimizing the root mean squared error can lead to good ranking performance, optimizing for the correct ordering within a query group may yield even better results, especially in a yearly context.

5.5 Summary

In conclusion, the results show that listwise ranking approaches (YetiRank Listwise, YetiRankPairwise Hybrid, and QueryRMSE Listwise) generally outperform pairwise ranking approaches and the traditional approach regress-then-rank for the ranking tasks evaluated. YetiRank Listwise, QueryRMSE listwise and YetiRankPairwise Hybrid are the top-performing methods, indicating the effectiveness of listwise approaches in the task of stock ranking model as they deal with a list of stocks. The outperformance of YetiRank and YetiRankPairwise Hybrid varoius when it comes to different time horizons. YetiRank outperformed in short investment horizons, and QueryRMSE outperformed in short-mid investment horizon. In addition,

YetiRankPairwise hybrid apprach ouperfomed in the longer investment horizon. At the end, listwise approaches have outperformes other approaches when it comes to stock ranking tasks.

Chapter 6

Conclusion

To conclude, LTR can be applied to portfolio optimization, where the goal is to identify the best combination of financial assets that can maximize return while minimizing risk. LTR can be used to rank a set of investment portfolios based on their expected returns and risk profiles, allowing investors to select the optimal portfolio based on their investment objectives and risk tolerance.

In addition to predicting market trends and identifying investment opportunities, LTR can also be used for risk management in financial markets. LTR can be used to rank a set of financial assets based on their exposure to different types of risks, such as market risk, credit risk, and liquidity risk, allowing investors to minimize their exposure to specific types of risks.

The performance comparison of different ranking approaches using various evaluation metrics led to several observations and conclusions. Regress-then-Rank (RMSE) was found to be outperformed by LTR algorithms for stock ranking, as it had the lowest performance overall. YetiRank (listwise), QueryRMSE (listwise), YetiRankPairwise (combination of listwise and pairwise) were found to be the most effective ranking approaches for the specific application. YetiRank outperformed all other algorithms and for both concentrated and diversified portfolios of stocks as it has highest scores for top 4, 8, 16, 32, and bottom 10 stocks. However, it is important to consider specific requirements and constraints, such as computational resources and data availability, when choosing a ranking approach.

The contribution of this research lies in its application of LTR algorithms leveraging GBM, as the predictive model, to the field of PER. The findings demonstrate that these methods can significantly improve the revenue of stock portfolios compared to using traditional regression-based approaches. This research has important implications for both theory and practice. The use of learning to rank methods has proven the improvement on the accuracy of stock ranking predictions, which is a critical component of financial decision-making. The findings could inform the development of new predictive models that utilize these methods to better forecast stock performance and form investment decisions. This study, on top of revenue generation, is a great asset to financial institutions as well as individual investors for portfolios risk management.

6.1 Future Work

In light of the findings and conclusions drawn from this study, there are several potential avenues for future research. Some possible directions for future work include:

- **Expanding the range of features:** Incorporating additional features, such as alternative data sources, macroeconomic indicators, or sentiment analysis from news articles and social media, on the performance of the ranking models.
- **Russell 3000 index:** Conducting a similar study on Russell 3000 index that consists of 3000 stocks is desirable for future work

By addressing these areas, future research can build on the findings of this study to further advance the field of machine learning-based stock ranking prediction and contribute to the development of more effective and robust investment strategies.

References

[1] D. Shah, H. Isah, and F. Zulkernine, "Stock market analysis: A review and taxonomy of prediction techniques," *International Journal of Financial Studies*, vol. 7, p. 26, May 2019.

[2] M. Adya and F. Collopy, "How effective are neural networks at forecasting and prediction? a review and evaluation," *J. Forecast.*, vol. 17, pp. 481–495, Sept. 1998.

[3] P. S. Rao, K. Srinivas, and A. K. Mohan, "A survey on stock market prediction using machine learning techniques," in *Lecture Notes in Electrical Engineering*, pp. 923–931, Springer Singapore, 2020.

[4] Á. Arnaiz-González, J.-F. Díez-Pastor, J. J. Rodríguez, and C. García-Osorio, "Study of data transformation techniques for adapting single-label prototype selection algorithms to multi-label learning," *Expert Syst. Appl.*, vol. 109, pp. 114–130, Nov. 2018.

[5] C.-L. Liu, B. Lovell, D. Tao, and M. Tistarelli, "Pattern recognition, part 1 [guest editors' introduction]," *IEEE Intell. Syst.*, vol. 31, pp. 6–8, Mar. 2016.

[6] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142, 2002.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, 2005.

[8] G. Preethi and B. Santhi, "Stock market forecasting techniques: A survey.," *Journal of Theoretical & Applied Information Technology*, vol. 46, no. 1, 2012.

[9] W. Huang, Y. Nakamori, and S.-Y. Wang, "Forecasting stock market movement direction with support vector machine," *Computers & operations research*, vol. 32, no. 10, pp. 2513–2522, 2005.

[10] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[11] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert systems with applications*, vol. 42, no. 1, pp. 259–268, 2015.

[12] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE transactions on neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[13] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] A. Borovykh, S. Bohte, and C. W. Oosterlee, “Conditional time series forecasting with convolutional neural networks,” *arXiv preprint arXiv:1703.04691*, 2017.

[16] S. Keshvari, F. Ensan, and H. S. Yazdi, “ListMAP: Listwise learning to rank as maximum a posteriori estimation,” *Information Processing &amp Management*, vol. 59, p. 102962, July 2022.

[17] M. Chen and X. Zhou, “DeepRank: Learning to rank with neural networks for recommendation,” *Knowl. Based Syst.*, vol. 209, p. 106478, Dec. 2020.

[18] S. Bruch, X. Wang, M. Bendersky, and M. Najork, “An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance,” in *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ACM, Sept. 2019.

[19] T. Thonet, Y. G. Cinar, E. Gaussier, M. Li, and J.-M. Renders, “Listwise learning to rank based on approximate rank indicators,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 8494–8502, June 2022.

[20] M. Ibrahim and M. Carman, “Comparing pointwise and listwise objective functions for random-forest-based learning-to-rank,” *ACM Trans. Inf. Syst.*, vol. 34, pp. 1–38, Sept. 2016.

[21] K. Hofmann, S. Whiteson, and M. de Rijke, “Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval,” *Inf. Retr. Boston.*, vol. 16, pp. 63–90, Feb. 2013.

[22] Y. Shi, M. Larson, and A. Hanjalic, "List-wise learning to rank with matrix factorization for collaborative filtering," in *Proceedings of the fourth ACM conference on Recommender systems*, ACM, Sept. 2010.

[23] X. Zhu and D. Klabjan, "Listwise learning to rank by exploring unique ratings," 2020.

[24] "Hybrid learning to rank for financial event ranking," in *Proceedings of the 44th International ACM SIGIR Conference author* =.

[25] A. Agarwal, K. Takatsu, I. Zaitsev, and T. Joachims, "A general framework for counterfactual learning-to-rank," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, (New York, NY, USA), ACM, July 2019.

[26] A. Agarwal, K. Takatsu, I. Zaitsev, and T. Joachims, "A general framework for counterfactual learning-to-rank," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 5–14, 2019.

[27] A. A. Rezaei, J. Munoz, M. Jalili, and H. Khayyam, "Vital node identification in complex networks using a machine learning-based approach," *SSRN Electronic Journal*, 2022.

[28] S. Saha, J. Gao, and R. Gerlach, "Stock ranking prediction using list-wise approach and node embedding technique," *IEEE Access*, vol. 9, pp. 88981–88996, 2021.

[29] Q. Song, A. Liu, and S. Y. Yang, "Stock portfolio selection using learning-to-rank algorithms with news sentiment," *Neurocomputing*, vol. 264, pp. 20–28, 2017. Machine learning in finance.

[30] X. Zhang and Y. Tan, "Deep stock ranker: A lstm neural network model for stock selection," in *Data Mining and Big Data* (Y. Tan, Y. Shi, and Q. Tang, eds.), (Cham), pp. 614–623, Springer International Publishing, 2018.

[31] X. Zhang, L. Wu, and Z. Chen, "Constructing long-short stock portfolio with a new listwise learn-to-rank algorithm," *Quantitative Finance*, vol. 22, pp. 321 – 331, 2021.

[32] T. Ma and Y. Tan, "Stock ranking with multi-task learning," *Expert Systems with Applications*, vol. 199, p. 116886, 2022.

[33] D. Poh, B. Lim, S. Zohren, and S. Roberts, "Building cross-sectional systematic strategies by learning to rank," *The Journal of Financial Data Science*, vol. 3, pp. 70–86, Mar. 2021.

[34] E. Tas and A. H. Atli, "Stock price ranking by learning pairwise preferences," *Computational Economics*, Nov. 2022.

[35] M. Alsulmi, "From ranking search results to managing investment portfolios: Exploring rank-based approaches for portfolio stock selection," *Electronics (Basel)*, vol. 11, p. 4019, Dec. 2022.

[36] G. Guo, Y. Rao, F. Zhu, and F. Xu, "Innovative deep matching algorithm for stock portfolio selection using deep stock profiles," *PLoS One*, vol. 15, p. e0241573, Nov. 2020.

[37] F. K. Reilly, K. C. Brown, and S. J. Leeds, *Investment analysis and portfolio management*. South-Western College, May 2018.

[38] J. Hair, R. Anderson, B. Babin, and W. Black, *Multivariate Data Analysis*. Andover, England: Cengage Learning EMEA, 8 ed., May 2018.

[39] C. F. Tsai and S. P. Wang, "Stock price forecasting by hybrid machine learning techniques," *Applied Soft Computing*, vol. 9, no. 2, pp. 567–573, 2009.

[40] A. Sharma, D. Bhuriya, and U. Singh, "Survey of stock market prediction using machine learning approach," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, IEEE, Apr. 2017.

[41] C. D. Kirkpatrick and J. R. Dahlquist, *Technical analysis: The complete resource for financial market technicians*. FT Press, 2010.

[42] J. D. Kelleher, B. Mac Namee, and A. Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. MIT Press, 2015.

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[44] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Stat.*, vol. 29, pp. 1189–1232, Oct. 2001.

[45] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," June 2017.

[46] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Front. Neurorobot.*, vol. 7, p. 21, Dec. 2013.

[47] D. Cossock and T. Zhang, “Subset ranking using regression,” in *Learning Theory*, Lecture notes in computer science, pp. 605–619, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[48] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, “An efficient boosting algorithm for combining preferences,” *Journal of Machine Learning Research*, vol. 4, pp. 933–969, 2003.

[49] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, “Learning to rank: From pairwise approach to listwise approach,” pp. 129–136, 2007.

[50] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, “Listwise approach to learning to rank - theorem and algorithm,” pp. 1192–1199, 2008.

[51] T.-Y. Liu, “Learning to rank for information retrieval,” *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2007.

[52] A. Kurennoy, J. Coleman, I. Harris, A. Lynch, O. Mac Fhearai, and D. Tsatsoulis, “A general framework for pairwise unbiased learning to rank,” in *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*, pp. 204–213, 2022.

Appendix

Implementation

Extracting Features

Data was collected by scraping the web and concatenating all 500 companies' market data. More features were then generated by calculations made on the existing publicly available data, where other features were added on top of the aforementioned. The following code shows the function used to calculate the features. The `compute_derived_features` function calculates a comprehensive set of technical indicators and derived features for each stock in the given DataFrame. The function first groups the data by the stock ticker and then computes the following features for each group

```
def compute_derived_features(df):
    grouped = df.groupby("Ticker")
    features = pd.DataFrame()
    for ticker, groupe in grouped:
        groupe = groupe.sort_values(by='Date')
        #1. RSI
        groupe['rsi'] = ta.rsi(groupe['Adj Close'])
        #2. Moving Average Convergence Divergence (MACD) -
           Momentum
```

```
        groupe['26_ema'] = groupe['Adj Close'].ewm(span=26,
            adjust=False).mean()
        groupe['12_ema'] = groupe['Adj Close'].ewm(span=12,
            adjust=False).mean()
        groupe['macd'] = groupe['12_ema'] - groupe['26_ema']
        groupe['signal'] = groupe['macd'].ewm(span=9, adjust=
            False).mean()
        groupe['hist'] = groupe['macd'] - groupe['signal']
        #3. Momentum Indicator - Momentum
        groupe['momentum_14'] = ta.mom(groupe['Adj Close'],
            length=14)
        #4. Rate of Change (ROC) - Momentum
        groupe['roc_14'] = ta.roc(groupe['Adj Close'], length
            =14)
        #5. Ultimate Oscillator (uo) - Momentum
        groupe['uo'] = ta.uo(groupe['High'], groupe['Low'],
            groupe['Close'], length=14)
        #6. Williams %R (willr) - Momentum
        groupe['willr'] = ta.willr(groupe['High'], groupe['Low'
            ], groupe['Close'], length=14)
        #7. Simple Moving Average (SMA) - Overlap
        groupe['sma_14'] = ta.sma(groupe['Adj Close'], length
            =14)
        #8. High-Low Average: hl2 - Overlap
        groupe['high_low_average'] = ta.hl2(groupe['High'],
            groupe['Low'])
```

```
        #9. Calculate the moving average
        groupe['ma5'] = groupe['Adj Close'].rolling(window=5).
           mean()
        groupe['ma10'] = groupe['Adj Close'].rolling(window=10)
           .mean()
        groupe['ma20'] = groupe['Adj Close'].rolling(window=20)
           .mean()
        groupe['ma50'] = groupe['Adj Close'].rolling(window=50)
           .mean()
        groupe['ma100'] = groupe['Adj Close'].rolling(window
           =100).mean()
        groupe['ma200'] = groupe['Adj Close'].rolling(window
           =200).mean()
        #10. Calculate the exponential moving average
        groupe['ema5'] = groupe['Adj Close'].ewm(span=5, adjust
           =False).mean()
        groupe['ema10'] = groupe['Adj Close'].ewm(span=10,
           adjust=False).mean()
        groupe['ema20'] = groupe['Adj Close'].ewm(span=20,
           adjust=False).mean()
        groupe['ema50'] = groupe['Adj Close'].ewm(span=50,
           adjust=False).mean()
        groupe['ema100'] = groupe['Adj Close'].ewm(span=100,
           adjust=False).mean()
        groupe['ema200'] = groupe['Adj Close'].ewm(span=200,
           adjust=False).mean()
```

```
        #11. Bollinger Bands (BB) - Volatility
        groupe['rolling_std'] = groupe['Adj Close'].rolling(
            window=5).std()
        groupe['upper_band'] = groupe['ma5'] + 2 * groupe['
            rolling_std']
        groupe['lower_band'] = groupe['ma5'] - 2 * groupe['
            rolling_std']
        #12. Calculate the daily volatility
        volatility = groupe['Adj Close'].rolling(window=30).std
            ()
        groupe['volatility'] = volatility
        #13. Average True Range (ATR) - Volatility
        groupe['atr'] = ta.atr(groupe['High'], groupe['Low'],
            groupe['Close'], length=14)
        #14. Average Directional Movement Index: adx - Trend
        groupe['adx_14'] = ta.adx(groupe['High'], groupe['Low'
            ], groupe['Close'], length=14)['ADX_14']
        #15. Kurtosis: kurtosis - Statistics
        groupe['kurtosis'] = ta.kurtosis(groupe['Adj Close'])
        #16. Skew: skew - Statistics
        groupe['skew_30'] = ta.skew(groupe['Adj Close'], length
            =30)
        #17. Standard Deviation: stdev - Statistics
        groupe['std'] = ta.stdev(groupe['Adj Close'], length
            =30)
        #18. Variance: variance - Statistics
```

```
        groupe['variance'] = ta.variance(groupe['Adj Close'],
            length=30)
        #19. Z Score: zscore - Statistics
        groupe['zscore'] = ta.zscore(groupe['Adj Close'])
        #20. On-Balance Volume: obv - Volume
        groupe['obv'] = ta.obv(groupe['Adj Close'], groupe['
            Volume'])
        #21. Money Flow Index (MFI) - Volume
        groupe['mfi'] = ta.mfi(groupe['High'], groupe['Low'],
            groupe['Close'], groupe['Volume'], length=14)
        #22. Draw Down: drawdown - Performance
        groupe['drawdown'] = ta.drawdown(groupe['Close'])['DD']
        #23. Log Return: log_return - Performance
        groupe['log_return_14'] = ta.log_return(groupe['Adj
            Close'], length=14)
        features = pd.concat([features, groupe], axis=0)
    return features
```

Once all the features are calculated for each stock group, the function concatenates these groups into a single DataFrame and returns the result. This DataFrame will contain the derived features for all stocks in the input dataset, which can then be used for further analysis, model training, or evaluation.

Dataset Optimization

The `optimize_dtypes` function is designed to optimize the data types of columns in a given pandas DataFrame to save memory and improve computational efficiency, without compromising the data's precision. The function takes several optional parameters that allow for fine-

grained control over the optimization process:

1. `max_categories`: Maximum number of unique values for a column to be considered categorical.
2. `float_from` and `float_to`: Convert float columns with precision `float_from` to precision `float_to`.
3. `float_to_int`: If set to True, convert float columns with no fractional part to integer data type.
4. `exclude_columns`: A sequence of column names to exclude from the optimization process.
5. `verbose`: If set to True, print additional information during the optimization process.

The function performs the following steps:

1. Initialize dictionaries and sets to store information about the data types and which columns to optimize.
2. Identify categorical columns with a low number of unique values and set their data type to 'category'.
3. Identify float columns with no fractional part and convert them to integer data type if possible.
4. Determine which integer columns can be converted to a smaller, more memory-efficient data type without losing any information.
5. Apply the optimized data types to the input DataFrame and return the result.

```
from typing import *
```

```
def optimize_dtypes(
    df: pd.DataFrame,
    max_categories: Optional[int] = 100,
    float_from: Optional[int] = None,
    float_to: Optional[int] = None,
    float_to_int: bool = True,
    exclude_columns: Sequence = (),
    verbose: bool = False,
) -> pd.DataFrame:
    """Compress datatypes in a pandas dataframe to save space
       while keeping precision."""



    '''
    Overall, this code is used to gather information about each
        column's data type and to identify the columns that
       should be optimized for
    memory usage by converting them to a more efficient data
       type. The dictionaries and sets created by this code
       will be used later in the
    optimize_dtypes function to perform the actual data type
       optimization.
    '''
    old_dtypes = {}
    float_dtypes = {}
    int_fields = set()
```

```
unchanged_dtypes = {}
for field, the_type in df.dtypes.to_dict().items():
    if field not in exclude_columns:
        old_dtypes[field] = the_type.name
        if "int" in the_type.name and the_type.name != "
           int8":
            int_fields.add(field)
        else:
            if float_from and float_to:
                if f"float{float_from}" in the_type.name:
                    float_dtypes[field] = f"float{float_to}
                       "
    else:
        unchanged_dtypes[field] = the_type.name
# ----------------------------------
# Every object var with too few categories must become a
   Category
# ----------------------------------
'''
Overall, this code identifies categorical columns in the
   input DataFrame and sets their data type to "category"
   if they have a low number
of unique values. This can help reduce memory usage and
   speed up subsequent computations.
'''
cat_dtypes = dict()
```

```
    if max_categories is not None:
        for col, the_type in old_dtypes.items():
            if "object" in the_type:
                try:
                    n = df[col].nunique()
                    if n <= max_categories:
                        cat_dtypes[col] = "category"
                except:
                    unchanged_dtypes[col] = the_type  # to
                       avoid stumbling on lists like [1]
    # ----------------------------------------
    # Checks for each float if it has no fractional digits and
       NaNs, and, therefore, can be made an int
    # ----------------------------------------
    '''
    Overall, this code block identifies float columns in the
       input DataFrame where all values have a fractional part
       of 0.0, and converts them to
    integer data type if possible. This can help reduce memory
       usage and speed up subsequent computations.
    '''
    if float_to_int:
        iterable = old_dtypes.items()
        if verbose:
            iterable = tqdmu(iterable, desc="Checking float2int
               ")
```

```
    for col, the_type in iterable:
        if "float" in the_type:
            fract_part, _ = np.modf(df[col])
            if (fract_part == 0.0).all():
                if not (df[col].isna().any()):  # NAs can't
                    be converted to int
                    if verbose:
                        logger.info("%s %s->int", col,
                            the_type)
                    int_fields.add(col)

# ----------------------------------------
# Finds minimal size of int suitable to hold each variable
   of interest without loss of coverage
# ----------------------------------------
'''
Overall, this code determines which integer columns can be
   converted to a smaller, more memory-efficient data type
   without losing any information.
The resulting int_dtypes dictionary will be used later in
   the optimize_dtypes function to actually perform the
   data type optimizations.
'''
int_dtypes = {}
if len(int_fields) > 0:
```

```
        max_vals = df[int_fields].max()
        min_vals = df[int_fields].min()

        int_powers = [8, 16, 32, 64]
        int_topvals = [np.iinfo("int"+str(p)) for p in
           int_powers]

        min_max = pd.concat([min_vals, max_vals], axis=1)
        min_max.columns = ["min", "max"]

        for r in min_max.itertuples():
            cur_p = int(old_dtypes[r.Index].replace("uint", "")
               .replace("int", "").replace("float", ""))
            for j, p in enumerate(int_powers):
                if p >= cur_p:
                    break
                if r.max < int_topvals[j].max and r.min >
                   int_topvals[j].min:
                    if verbose:
                        logger.info("%s [%s]->[%s]", r.Index,
                           old_dtypes[r.Index], p)
                    int_dtypes[r.Index] = "int" + str(p)
                    break

    # ----------------------------------------
    # Actual converting & reporting.
```

```
    # ----------------------------------------
    '''
    Overall, this code applies the optimized data types to the
       input DataFrame and returns the result. This can help
       reduce the memory footprint
    of the DataFrame and speed up subsequent computations.
    '''
    new_dtypes = {**cat_dtypes, **float_dtypes, **int_dtypes,
       **unchanged_dtypes}
    if len(new_dtypes) > 0:
        if verbose:
            logger.info(f"Going to use the following new dtypes
               : {new_dtypes}")
        return df.astype(new_dtypes)
    else:
        return df
```

By calling the `optimize_dtypes` function with the `float_from=64` and `float_to=32` parameters, it is requesting to convert all float64 columns in the `df_selected` DataFrame to `float32` data type, if possible. Getting rid of FP64 is important in machine learning applications where memory usage and computational complexity are concerns, as it requires twice as much memory as FP32 and can lead to slower training times. Using FP32 instead can provide a good balance between precision and efficiency.

```
df_selected=optimize_dtypes(df_selected,float_from=64, float_to
   =32)
```

The resulting DataFrame, `df_selected`, will have its `float64` columns converted to `float32`,

and any other applicable optimizations will also be applied. By optimizing the data types, the function can reduce the memory footprint of the DataFrame and speed up subsequent computations.

Target Variable Calculation

This code defines two functions, `get_future_price` and `compute_stock_future_price`, and then applies them to the `df_selected` DataFrame to compute the 1-month future price and return for each stock.

1. `get_future_price`: This function takes a single group of stock data as input and sorts it by date. It then computes the future price by shifting the `Adj Close` column back by `DAYS_TO_LOOK_AHEAD` (21 weekdays) and returns the group with the new `future_price` column.

2. After defining the functions, `compute_stock_future_price` is called with the `df_selected` DataFrame to compute the future price for each stock.

3. The DataFrame `df_target` is sorted by 'Date' and 'Ticker' in ascending order.

4. The `future_price` column is extracted from the `df_target` DataFrame and stored in the `future_price` variable. The `future_price` column is then deleted from the `df_target` DataFrame.

5. Rows with NaN values in the `future_price` column are removed from both `df_target` and `future_price`.

6. Finally, the 1-month return is calculated as the percentage difference between the future price and the adjusted close price, and the result is stored in the `stock_return_1month` variable as a `float32` data type.

```
DAYS_TO_LOOK_AHEAD=21 # 1 financial month

def get_future_price(groupe):
    groupe = groupe.sort_values(by='Date')
    groupe['future_price'] = groupe['Adj Close'].shift(-1*
       DAYS_TO_LOOK_AHEAD)
    return groupe

def compute_stock_future_price(df):
  grouped_df = df_selected.groupby('Ticker')
  result = pd.concat([get_future_price(group) for _, group in
     grouped_df])
  return result

df_target = compute_stock_future_price(df_selected)
df_target.sort_values(by=["Date","Ticker"], ascending=[True,
   True],inplace=True)

future_price=df_target['future_price']
del df_target['future_price']

idx=~future_price.isna()
df_target=df_target[idx]
future_price=future_price[idx]
stock_return_1month=( 100.0*(future_price-df_target['Adj Close'
   ])/df_target['Adj Close']).astype(np.float32)
```

Data Preparation

The `create_dataset` function takes a pandas DataFrame df and a list of dates "days" as input parameters. The purpose of this function is to create a dataset with a fixed number of stocks per day from the input DataFrame. The function returns a tuple containing the selected stock data (`X`) and their corresponding relevance scores (`y`).

```
def create_dataset(df:pd.DataFrame,days:list, concat:bool=True)
   ->tuple:
  X = [];y = []
  for query in tqdm(days,desc="Days",leave=False):
    idx=df.index==query

    all_daily_stocks = df[idx]
    if len(all_daily_stocks)>=NSTOCKS_PER_DAY:
      #randomly select desired number of stocks
      stocks_indices=np.random.choice(np.arange(len(
         all_daily_stocks)),size=NSTOCKS_PER_DAY,replace=False)
      #stocks_indices=np.arange(len(all_daily_stocks))
      relevance_scores =stock_return_1month[idx].iloc[
         stocks_indices]
      min_score,max_score=relevance_scores.min(),
         relevance_scores.max()

      relevance_scores=(relevance_scores-min_score)/(max_score-
         min_score)
```

```
      X.append(all_daily_stocks.iloc[stocks_indices])
      y.append(relevance_scores)
  if concat:
    return pd.concat(X),pd.concat(y)
  else:
    return X,y
```

Model Functions

fit_GBM_model()

This function trains a `CatBoostRanker` model on the input training data (`train_pool`) and validates the model on the input validation data (`val_pool`). It takes in a `loss_function` argument, which is the objective function to optimize during training, and an optional `additional_params` dictionary containing additional parameters to pass to the `CatBoostRanker` model. It returns the trained model.

```
def fit_\ac{GBM}_model(loss_function, train_pool, val_pool,
   additional_params=None):
    parameters = deepcopy(default_parameters)
    parameters['loss_function'] = loss_function
    parameters['train_dir'] = loss_function

    if additional_params is not None:
        parameters.update(additional_params)

    model = CatBoostRanker( **parameters)
```

```
    model.fit(train_pool, eval_set=val_pool, plot=True)

    return model
```

prepare_df_for_catboost()

This function prepares the DataFrame for use with the `CatBoost` library. It replaces missing values (NAs) with a specified string value and converts categorical variables to the appropriate data type. It takes in a DataFrame object, `columns_to_drop` (a list of columns to drop), `text_features` (a list of textual features to process), `cat_features` (a list of categorical features to process), and `na_filler` (a string value to replace NAs with). It modifies the input DataFrame in place.

```
def prepare_df_for_catboost(df: object, columns_to_drop: list =
    [], text_features: list = [], cat_features: list = [],
   na_filler: str = "") -> None:
    """
        Catboost needs NAs replaced by a string value.
        Possibly extends cat_features list.
    """
    cols = set(df.columns)

    for var in tqdmu(text_features, desc="Processing textual
       features for CatBoost..."):
        if var in cols:
            if var not in columns_to_drop:
                df[var] = df[var].fillna(na_filler)
```

```
    for var in tqdmu(cols, desc="Processing categorical
        features for CatBoost..."):
        if var in cols:
            if isinstance(df[var].dtype, pd.CategoricalDtype):
                if "" not in df[var].cat.categories:
                    df[var].cat.add_categories(na_filler) #,
                        inplace=True
                    df[var] = df[var].astype(str)
                df[var] = df[var].fillna("")
                if var not in cat_features:
                    print(f"{var} appended to cat_features")
                    df[var] = df[var].astype(str)
                    cat_features.append(var)

            else:
                if var in cat_features:
                    df[var] = df[var].fillna(na_filler).astype(
                        str)
```

`create_pool`

This function evaluates a trained GBM model on multiple validation pools. It takes in a trained model and a list of pools, where each pool is a tuple containing the pool object, `true_values` (the true target labels), `group_id` (a unique identifier for each group of samples), and `pool_name` (a string that identifies the pool). It calls `evaluate_predictions()` to evaluate the model predictions and print the evaluation results.

```
def create_pool(X,y,cat_features:list)->tuple:
  if isinstance(X,list):
    X,y=pd.concat(X),pd.concat(y)
  group_id=X.index.values.astype(str)
  pool = Pool(
      data=X,
      label=y,
      group_id=group_id,
      cat_features=cat_features
  )
  return pool,group_id
```

evaluate_trained_GBM_model()

This function prepares the DataFrame for use with the `CatBoost` library. It replaces missing values (NAs) with a specified string value and converts categorical variables to the appropriate data type. It takes in a DataFrame object, `columns_to_drop` (a list of columns to drop), `text_features` (a list of textual features to process), `cat_features` (a list of categorical features to process), and `na_filler` (a string value to replace NAs with). It modifies the input DataFrame in place.

```
def evaluate_trained_\ac{GBM}_model(model,pools):
  for pool_name,pool,true_values,group_id in tqdm(pools):
    predictions=model.predict(pool)
    evaluate_predictions(predictions=predictions,true_values=
       true_values,group_id=group_id,pool_name=pool_name)
```

`evaluate_predictions()`

This function evaluates a set of predictions made by a GBM model on a given validation pool. It takes in the predictions, `true_values`, `group_id`, and `pool_name` as arguments. It evaluates the predictions using a list of predefined metrics (`LTR_Metrics`) and prints the results.

```
def evaluate_predictions(predictions,true_values,group_id,
   pool_name:str,):
    print(f"{pool_name} set, predictions_min={predictions.min()
       }, predictions_max={predictions.max()}")
    if isinstance(true_values,pd.Series):
      true_values=true_values.values
    for metric in \ac{LTR}_Metrics:
      res=eval_metric(true_values, predictions, group_id=
         group_id,metric=metric)[0]
      print(f"\t{metric}={res}")
    print(f"\trbo_score={rbo_score(pred=predictions,y=
       true_values)}")
```

Model's Parameters

The split on the dataset is equivalent to the following: train (0.8), validation (0.1), test (0.1). In addition, the model parameters given the dataset and the task of stock ranking are as follows:

- **nan_mode:** Determines how to handle missing values in the input data. The 'Min' setting replaces missing values with the minimum value for the feature.
- **gpu_ram_part:** Sets the fraction of GPU memory to be used by the model. In this case, 95% (0.95) of the GPU memory will be used.

- **eval_metric:** Specifies the evaluation metric to be used during model training. The 'QueryRMSE' metric is used here, which is suitable for ranking problems.
- **iterations:** The number of boosting rounds (trees) to be built. The model will train for 1000 iterations.
- **leaf_estimation_method:** The method used for leaf estimation. The 'Newton' method is used here.
- **grow_policy:** Determines the tree growth policy. 'SymmetricTree' policy is used, which enforces equal depth for all leaves.
- **boosting_type:** Specifies the boosting method to be used. The 'Plain' method is used in this case.
- **one_hot_max_size:** Maximum number of categories for one-hot encoding. If a categorical feature has fewer categories than this value, it will be one-hot encoded.
- **l2_leaf_reg:** Coefficient at the L2 regularization term of the cost function. In this case, the value is set to 3.
- **random_strength:** The amount of randomness to use for scoring splits when building the trees. A value of 1 is used here.
- **depth:** The maximum depth of the trees that will be built. In this case, the value is set to 6.
- **learning_rate:** The learning rate used for boosting. In this case, the value is set to 0.03.
- **task_type:** Specifies the type of device to be used for training. 'GPU' is used here, which means the model will be trained on a GPU.
- **bootstrap_type:** Specifies the sampling method for the training set. 'Bayesian' is used here, which means that the model will use Bayesian bootstrapping.

Summary

This implementation chapter covers the process of collecting and preparing data for a machine learning model using web scraping and feature generation techniques. It includes the calculation of derived features for stocks, optimization of dataset memory usage, calculation of target variables, and data preparation using various functions. Additionally, several model functions are provided for training and evaluating `CatBoostRanker` models, streamlining the entire process of building and testing machine learning models on the given data.