Arthur GUIMARD,
Alexandru VERDES,
Aharon ELBEZ

ENSAE third year
Machine Learning Project
Year 2023 - 2024

Systematic Stock Selection Strategy using Learning-to-Rank algorithms

ENSAE
Paris, France

Summary

1 Introduction

Portfolio Optimization is an issue at stake for investors as it's offering a means to mitigate risk and ensure rentability. A common approach is to rank a specific set of securities based on well-specified constraints.

Originating in the early 2000s, learning-to-rank concepts were first and foremost used to construct search engines (Like Google). For instance, the RankNet algorithm, introduced by Microsoft Research in 2005, marked a significant step forward during this period.

In the last decade, learning-to-rank methods have considerably progressed, making them applicable to portfolio optimization problems. This paper, titled **"Building Cross-Sectional Systematic Strategies By Learning to Rank,"** explores an innovative ranking method for cross-sectional systemic strategies. The core objective is to improve the performance of the ranking process, thereby enhancing portfolio optimization. The algorithms employed in this project are the **LambdaMART** and the **YetiRank** methods.

In our project, we introduce the learning-to-rank concept, the LambdaMART and the YetiRank methods in Section 2. Section 3 focuses on the implementation of these algorithms and a Naive Implementation Case. We end Section 3 by discussing the results of the models and optimizing the parameters. Lastly, we build a systematic trading strategy using our trained models and discuss the relevance of the methods for this purpose.

2 Learning to Rank and LambdaMART

2.1 Learning to Rank Theory

In this paper, ranking is defined as a classification of documents based on their relevance to a specific query. In ranking models, the goal is to predict pertinence scores as the *target variable* of the model for each input $x = (q, f)$, where q is a query and f is a document including a list of characteristics. Therefore, the predicted pertinence scores allow the user to rank the documents given a query. The Learning To Rank model has a training phase with a loss minimization problem:

$$\min_{x \in \mathcal{X}} f(x)$$

where $\mathcal{X}$ is the whole set of queries and documents.

The main issue at stake for a ML model is to determine the correct loss function. There are three types of loss functions commonly used in Learning to rank models : Pointwise, Pairwise and Listwise.

2.2 RankNet and LambdaRank

2.2.1 RankNet approach

RankNet is a pairwise Learning to Rank model, which is the foundation for more recent models such as LambdaRank and LambdaMART.

Let a dataset $(x_i, q_i)_i^n \in \mathcal{D} \times \mathcal{Q}$, where $\mathcal{D}$ represents the set of documents and $\mathcal{Q}$ is the set of queries. For each pair (x_i, x_j), RankNet computes a score $s_i = f(x_i)$. The model aims to predict the probability that x_i has a greater rank than x_j for $i \neq j$, given a specific query q. RankNet uses a sigmoid function for its pairwise approach:

$$P_{i,j} = \mathbb{P}[x_i \succ x_j] = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

Here, σ is an arbitrary constant. The next step involves computing an error term to measure the difference between true and predicted probabilities. RankNet utilizes a cross-entropy cost function:

$$C_{i,j} = -\overline{P}_{i,j} \log(P_{i,j}) - (1 - \overline{P}_{i,j}) \log(1 - P_{i,j})$$

The actual probability $\overline{P}_{i,j}$ takes its values in $\{0, \frac{1}{2}, 1\}$, where $\overline{P}_{i,j} = \frac{1}{2}(1 + T_{i,j})$ and $T_{i,j} \in \{-1, 0, 1\}$. Simplifying the equation, we get:

$$C_{i,j} = \frac{1}{2}(1 - T_{i,j})\sigma(s_i - s_j) + \log(1 + e^{-\sigma(s_i - s_j)})$$

The optimization of the RankNet loss function employs stochastic gradient descent. Let θ_t be the model parameters, and γ_t be the learning rate:

$$\theta_{t+1} = \theta_t - \gamma \left(\frac{\partial C_{i,j}}{\partial s_i} \frac{\partial s_i}{\partial \theta_t} + \frac{\partial C_{i,j}}{\partial s_j} \frac{\partial s_j}{\partial \theta_t} \right)$$

However, to expedite this process, RankNet introduces a speed optimization:

$$\frac{\partial C_{i,j}}{\partial \theta_t} = \lambda_{i,j} \left(\frac{\partial s_i}{\partial \theta_t} - \frac{\partial s_j}{\partial \theta_t} \right)$$

where

$$\lambda_{i,j} = \frac{\partial C_{i,j}}{\partial s_i} = \sigma \left(\frac{1}{2}(1 - T_{i,j}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right)$$

This optimization method involves accumulating λ for each document within a query, summing contributions from all pairs (x_i, x_j), and updating the model parameters.

2.2.2 LambdaRank approach

The λ we introduced in the previous part is the key concept involved in the LambdaRank approach.

In the course of RankNet training, the specific costs themselves are not required; what holds significance are the gradients (λ) of the cost concerning the model score. Picture these gradients as small arrows linked to each document in the ordered list, conveying the preferred direction for the adjustment of those documents. In fact, if x_j is more relevant than x_i,then x_i will receive a downward force with a magnitude of $|\lambda$, and simultaneously, x_j will undergo an equal and opposite upward force.

Figure 1: Caption

In theory, LambdaRank assumes that we can hypothesize an implied cost function $\overline{C}_{i,j}$ such that :

$$\lambda_{i,j} = \frac{\partial \overline{C}_{i,j}}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta_{NDCG}|$$

where $|\Delta_{NDCG}|$ is the change of the size in NDCG which is a metric we are going to define in the last part of this section. It can be observed that the implied cost function computed by LambdaRank is a product between the binary cross entropy cross function and the change of the size in NDCG. Indeed, we cannot always determine λ as gradients of a certain function only under certains conditions and LambdaRank assumes that this product corresponds to theses conditions. Lastly, the λ that are computed for each pear (x_i, x_j) for a given query are then summed and allow the update of the gradient descent.

2.3 LambdaMART and metrics evaluation

2.4 MART and LambdaMART

Finally , LambdaMART is a mix between LambdaRank and MART and has recently showed its greater efficiency.

Multiple Additive Regression Trees (MART) is a class of algorithm that can be trained on different models (classification, regression and ranking problems). MART, as an ensemble technique, combines forecasts from a multitude of decision trees. The construction of each tree

is carried out sequentially with the objective of rectifying the errors present in the ensemble.

MART uses a boosting strategy, wherein each successive tree addresses the residuals of the ensemble. The gradient of the objective function concerning predicted values is calculated, directing the construction of new trees to minimize this gradient.

This is where the LambdaRank and the MART complement each other as the LambdaRank focuses on the λ defined earlier which are gradients. Evidently, the synergy between MART and LambdaRank is apparent, given that MART inherently models gradients, and LambdaRank operates by explicitly specifying gradients at various points during training.

2.5 YetiRank

Just like LambdaMart, YetiRank is a Learning to Rank algorithm using a pairwise approach. It is known to work well with specific ranking metrics such as NDCG (a metric that we will use in our implementation).

2.6 Metrics for ranking model

As a crucial aspect in addressing machine learning challenges, it is imperative to evaluate ranking models using specific metrics. In this document, we explore one prominent metric: Normalized Discounted Cumulative Gain (NDCG). This metric will be used to assess the performance of our model. For clarity, the notation $@K$ signifies a focus on obtaining accurate rankings exclusively for the best K documents. This notation is particularly relevant in ranking scenarios where precision for only the top K documents is emphasized. This distinction is fundamental because, for instance, $NDCG@K$ yields different results compared to NDCG. Moreover, the systematic strategy built in this project only relies on the correct relevance of the top rankings.

Normalized Discounted Cumulative Gain (NDCG) is a ratio that aims to measure the quality of the ranking model. It is defined as follows:

$$NDCG@K = \frac{DCG@K}{IDCG@K}$$

Here, $DCG@K$ is the Discounted Cumulative Gain, and $IDCG@K$ is the Ideal Discounted Cumulative Gain.

$$DCG = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)}$$

In this formula, rel_i represents the relevance of the item at position i, and n is the total number of items. The fundamental concept of NDCG involves comparing the computed ranked list with the ideal (actual) ranked list.

The NDCG is an interesting metric in our case because of the superior weighting it gives to the most relevant documents. Indeed, in the strategy described at the end of the report, the models will be applied to only select the top ranked stocks.

3 Stock Selection Systematic Strategy using Learning-to-Rank algorithms

3.1 Description of the strategy

The strategy is based on the use of Learning to Rank algorithms to select the best stocks to buy at a certain frequency (here monthly).
Each month, the trained model is given a new set of inputs (features) with the most recent information about the stock's performance. The ranker model now identifies the 35 best stocks. For the entire upcoming month, a long position (meaning we profit from an increasing in stock prices) is taken on the top stocks.

3.2 Why use Learning to Rank models for stock selection ?

When considering strategies relying on recurrent stocks selection, being able to rank the stocks effectively helps with choosing the assets producing the greatest outperformance and reducing the risks. Currently, what is most used in the industry is ranking stocks simply based on their most recent returns (as done in the Naive/Base Case Implementation described later) as it is considered to be the most accurate indicator of future performance. Other methods use a wider range of metrics taking into account historical volatility for example.
Through our implementation, we want to test if using Learning-To-Rank algorithms can lead to a better information retrieval regarding the considered stocks. We will consider pairwise algorithms, check their performance on financial data and see if they ultimately allow better results in the backtest of the strategy.

3.3 Modelisation framework

Universe of stocks : The list of available stocks to choose from at each rebalancing date (ranking date) is the components of the stock index S&P 500. The close prices of each stock has been extracted through the Yahoo Finance API.

(Query, Documents) : The query is represented by a rebalancing date, in our case, the first trading date of each month. For each date, the documents that the model ranks is the universe of stocks

Features used to describe the stocks :

1. Returns with the following lookback periods (1m, 3m, 6m, 12m)
2. Historical volatility with the following lookback periods (1m,3m, 6m, 12m)
3. Sharpe ratios with the following lookback periods (1m,3m, 6m, 12m)
4. MACD (Moving Average Convergence Divergence) indicator

3.4 Preprocessing and data engineering

The features are generated as follows :

Returns : $r^i_{zmonth,t} = \frac{ClosePrice^i_t}{ClosePrice^i_{t-20*z}} - 1$; with z in 1,3,6 and 12

Historical Volatility : $\sigma^i_{histo,z} = \sqrt{\frac{1}{z*20}\sum_{j=0}^{20z-1}(r_j - \overline{r})^2}$; with z in 1,3,6 and 12

Sharpe Ratio : $Sharpe^i_z = \frac{r^i_{zmonth,t}}{\sigma^i_{histo,z}}$; with z in 1, 3, 6 and 12

MACD : $MACD = EMA_{12} - EMA_{26}$
with 26 and 12 period Exponential Moving Average (EMA_{26} and EMA_{12})

3.5 Models implementation

Train/Test split :

Train period : from January 2010 to December 2018
Test period : from December 2018 to January 2024

True Score/Rank :

A crucial part for the relevance of the strategy is the definition of a score or ranking system that matches our objective for stock selection. Since we have a monthly rebalancing frequency, we chose to build our score on the return 1 month.
Hence, $Score^i_m = r^i_{1month,m+1}$ with r_{1month} the return 1-month, i the index of the stock, and m the current month. The rank is then deduced by simply sorting the scores obtained for each monthly rebalacing date. With this choice of metric, if AAPL is the best performing stock in February, then its rank in January will be 1 in the training set.

3.5.1 Naive/Base Case Implementation

In order to assess the performance of our Learning-to-Rank algorithms, we need to have a base case for comparison. To serve this purpose, the predicted score will simply depend on the $r_{1month,t}$:
$PredScore^i_m = r^i_{1month,m}$
The predicted rank is deduced by sorting the predicted scores for each date.

3.5.2 LamdbaMART Implementation

To implement LambdaMART, we used Python's Lightgbm library. The package offers a LGBM-Ranker for LambdaMART implementation. Its computational efficiency enabled us to obtain reasonable model training times for our limited ressources. For the initial implementation, we considered the following parameters:

- Objective = lambdarank
- Boosting type = gbdt (using the Gradient Boosting Decision Tree)
- Number of estimators = 100
- Importance Type = gain
- metric= NDCG

- Number of leaves = 10
- Learning Rate = 0.1
- Max Depth = -1

We chose a low number of estimators (=100) primarily for computational reasons. Similarly for the learning rate, we started with a value of 0.1, which we will adjust if necessary later on. The evaluation metric considered for the model is the NDCG as introduced earlier.

3.5.3 YetiRank Implementation

To implement the YetiRank, we used the Catboost python library. Like LightGBM, it's a high-performance package which allowed the use of a YetiRank function. For the initial implementation, we considered the following parameters for reasons similar to those mentioned for LambdaMART:

- Loss Function : YetiRank
- Iterations : 200
- Depth : 5

3.5.4 Results

Metrics	Naive Case	LambdaMART	YetiRank
NDCG35	0.51	0.51	0.49

For the moment, the performance of the LambdaMART and YetiRank models is not very encouraging. Indeed, we can't see any outperformance of the models compared with the naive case.
One of the most important cause likely lies in our restricted number of features that only capture characteristics linked to the stocks prices. Indeed the financial markets involve a significant part of randomness that investors keep trying to model and predict. However, it is influenced by a wide variety of factors including market sentiment, general economic situation but also current news.

3.6 Model Fine Tuning

Due to the results, it has been decided to pursue only with the LambdaMART model and to try to tune its parameters to improve its performance.

To do so, a GridSearch has been implemented in order to test some different parameters. Since the Lightbgm library already uses parallelization in order to speed up the fit, the code is not written to use multiprocessing.

A set of different number of leaves, learning rate and number of estimators has been tested and the best performing set of parameters was the following :

- learning rate : 0.8
- number of estimators : 30
- number of leaves : 1000

This fine-tuning led to an improvement in the NDCG35 which was finally 0.536.

3.7 Strategy Backtest

The strategy (described in 3.1) is backtested as a form of index. The calculation starts with a base level of 1000. Then, each month, we determine the top performers (35 stocks) based on the ranking provided by the various methodologies we described above (Naive / LambdaMART / YetiRank). The group is selected in the portfolio for the entire month. We take a long position on the top group. When we reach the rebalancing date of the next month, we repeat the stock selection and keep cumulating the returns.

We also add volatility target mechanism through a simple exposure factor computed as follows :

$$E_t = min(E_{max}, \frac{VT}{VOL_{est.}(t)}) \tag{1}$$

with $E_{max} = 1$ the maximum exposure, $VT = 15\%$ the volatility target and $VOL_{est.}(t)$ the empirical estimated volatility of the top group

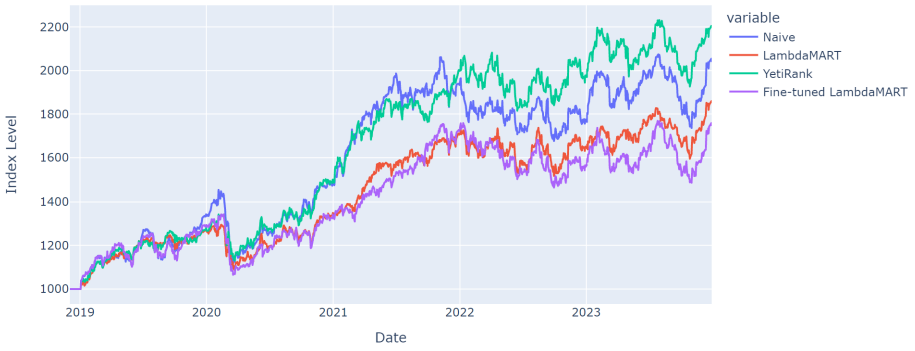
It will allow to have a fixed volatility of around 15% when comparing the different cases. The daily index levels are computed following this formula :

$$IL(t) = IL(t-1) * (1 + E_{t-1} * (\frac{1}{35} \sum_{i=1}^{35} r_t^i))$$

with IL(t) the Index Level of the strategy at date t, E_t the exposure at date t

3.7.1 Backtest results

The strategies have been backtested and the results are the following :



To compare the results, one can look at the following statistics :

	Annualized Return	Annualized Volatility	Sharpe Ratio
Naive	15.11	15.36	0.98
LambdaMART	12.96	13.41	0.97
YetiRank	16.87	14.49	1.16
Fine-tuned LambdaMART	11.66	15.12	0.77

We observe that the LambdaMART (even with optimized parameters) provides the lowest performance, even below the naive case. On the other hand, from a backtest point of view YetiRank surprisingly provides the best results in terms of annualized returns and sharpe ratio. This can be explained due to our monthly rebalancing frequency allowing for a drift in stock performance during the entire month.

4 Conclusion

The results of the implementation of Learning to Rank algorithms seem to be unsatisfactory in a context of building a systematic trading strategy solely based on the prices of the assets. Indeed, the features used for our LambdaMART and YetiRank algorithms are insufficient to provide a significant difference in the relevance identification of the stocks compared to the naive case.
This partially explains why such classic naive methods are still used in the industry as they provide comparable results to such implementations of state-of-the-art algorithms. Indeed, the naive approach provides a good trade off between performance and complexity/amount of data required.

In order to fully take advantage of Learning-to-Rank techniques, more complete datasets need to provided. For example, the list of features could be enriched with market sentiment indicators, economic metrics, or even financial data related to income statements.

References

[1] I. Lyzhin, A. Ustimenko, A. Gulin, L. Prokhorenkova *Which Tricks Are Important for Learning to Rank ?* https://arxiv.org/abs/2204.01500 (last revised on oct. 2023)

[2] D. Poh, B. Lim, S. Zohren, S. Roberts *Building Cross-Sectional Systematic Strategies By Learning to Rank* The Journal of Financial Data Science Spring (2021)

[3] C. Burges *From RankNet to LambdaRank to LambdaMART: An Overview* Microsoft Research Technical Report (2010)