Stock Ranking with Market Microstructure, Technical Indicator and News

Liang Wang  
University of Georgia  
Athens, GA, USA  
kavenvih@gmail.com

Khaled Rasheed  
University of Georgia  
Athens, GA, USA  
khaled@uga.edu

Abstract— Using machine learning techniques to assist financial decision making surged in several areas in the past decade. Text mining introduces count, tonality and sentiments of financial buzz into machine learned equity price models. The availability of high-frequency data enriches the forecasting models with features from market microstructure. In this research, we take a new perspective that directly learns the stocks’ relative performance with a ranking algorithm. We argue that the traditional regress-then-rank approach casts the portfolio selection practice into an unnecessarily hard problem and show that ranking algorithms outperform the neural network regressor significantly in terms of both ranking quality and simulated profit on out-of-sample testing data. With simulated trading under rigorous constraints of transaction costs and order execution price, we demonstrate that the ranker can be used to build highly profitable portfolios.

Keywords— Data Mining, Ranking, Stock Forecasting, News Analysis, High Frequency

I. INTRODUCTION

Using machine learning techniques to assist financial decision making surged in several areas in the past decade. Text mining introduces count, tonality and sentiments of financial buzz into machine learned equity price models [1]. Deep learning methods introduce extra layers of feature abstraction, such as trend extraction and public attentiveness detection, that mimics human decision-making process [2]. The availability of high-frequency data has driven researchers to explore data at finer granularity and examine the dynamic details about price formation [3]. These recent developments, together with many previously published works in financial forecasting, typically take a two-step regress-then-rank approach, which builds regression or classification models that predicts future returns, and then make investment suggestions from the stocks with higher predicted yield.

In this research, we take a new perspective that directly learns the stocks’ relative performance with a ranking algorithm. We argue that the traditional regress-then-rank approach casts the portfolio selection practice into an unnecessarily hard problem in the sense that traders typically pick their stocks without forming an accurate prediction of the target prices. LambdaRank is a group of ranking algorithms that have been proven successful in solving ranking problems on big data from the web [4]. It uses gradient descent learners, such as backpropagation neural networks [5] and MART [6], to model the labels that represent the relative order on a given set of instances. Since labels like this are inconsistent by nature, LambdaRank is designed to learn the probability that an instance should be ranked higher than another. A probabilistic model is built to optimize for an augmented cross entropy cost function that gives higher penalty to ranking mistakes on top performers. This skewed emphasis also fits naturally into the portfolio selection task, in the sense that traders care more about the accuracy on the predicted top performing stocks. We compare our approach with the traditional regress-then-rank approach by building a ranker and a neural network regressor on the same features. The result suggests that the ranker outperforms the neural network regressor significantly both in terms of ranking quality and simulated profit on out-of-sample testing data, and that the ranker can be used to build highly profitable portfolios after deduction of transaction costs.

Feature design is of great importance to the performance of a forecasting system. Ideally, the learning algorithm needs to get all factors that have impact on stock prices as features. Traditional traders make investment decisions based on past stock prices, fundamental variables [7], technical rules [8] and news. But since the invention of algorithmic traders, there is an increasing diversity in the factors that influence market participants’ decision making. An automated trader is capable of extracting features from high-frequency order flow and transaction data and derive trading strategies from it [9]. In order to empower our learners with features that may influence both human traders and algorithmic traders, we provide a wide spectrum of features to the learners, including past stock prices, technical indicators and rules, news and features to describe order flow and order book dynamics.

II. BACKGROUND AND RELATED WORKS

Using computer algorithms to forecast stock price, return, risk and their composites emerged vaguely after 1990. The mainstream of this topic can be clustered by the underlying trading practices, including fundamental analysis, technical analysis, trading with news and high frequency trading.

Fundamental analysts believe in the intrinsic value of equities and make relatively long-term forecasts based on metrics reported in financial statements and macroeconomic variables. Previous research used decision trees [10], neural

---

1 The ranking for a stock may float up or down due to changes in other stocks’ performance, while its own features remain constant.

Practitioners use rules built on top of technical indicators to assist their trading. Various indicators had been developed based on statistics of the past stock prices [8]. Most technical rules can be generalized as crossovers among the indicators, static thresholds and first or second order derivates of indicators and stock prices. Previous research used various machine learning techniques such as neural networks [15], genetic programming [16] and fuzzy logic [17] to predict the market with strong evidence of excess returns on out-of-sample test. Another branch of technical analysis is charting (a.k.a. pattern study). Many studies were conducted to examine the profitability of price patterns, such as head-and-shoulder [18], double-bottom [19] and rounding bottoms [20].

Financial text mining is a relatively new branch that did not get much attention until 1998 when Wüthrich et al. published their seminal paper [21]. Their system forecasted the direction of the DJIA index using articles collected from the Wall Street Journal and reported a simulated profit of around 7.5% over a three-month period. The wide spread of social media such as Twitter and StockTwits deepens the impact of textual information on stock markets [22, 23]. More recently, with the heated discussion about deep learning, several deep network structures were proposed for stock prediction with textual data [2, 24-26].

Driven by the accumulating evidence of decreased profitability from analyzing daily data [27, 28], considerable recent literature has switched to higher data frequency for forecasting opportunities [29, 30]. High frequency trading, however, is not just applying forecasting frameworks described above to data of higher frequency, but a developing research area that challenges existing theories and trading practices [9]. Many published works in this area attributed stock price changes to market events and microstructure, such as probability of order arrival [31], order flow imbalance [32], share volume of various order types [33] and gaps in order book [34]. These perspectives provide a more detailed picture of the dynamics in demand-supply and market’s reaction in terms of stock prices [35]. Using machine learning algorithms to model price changes with market microstructure features emerged recently [36], but published research under this topic is still very rare.

Stock ranking has been mentioned several times in the literature. Some used decision trees [37, 38], others used genetic programming [39] and neural networks [40]. However, existing literature either models the stock ranking indirectly (build models on stock return, then rank the stocks according to model predictions) or cast the ranking problem as an ordinal regression problem [37], which complicates the original problem even more than regress-then-rank. In contrast, ranking algorithms, such as RankNet [41], LambdaRank [42] and LambdaMART [4], learn the ranking among instances directly. These algorithms have been proved successful in information retrieval, but there is very limited published work that applied ranking algorithms for portfolio building, if any.

Our research extends the current literature by applying LambdaMART to stock ranking with features from market microstructure, news, past stock price and technical analysis.

### III. METHODOLOGY

LambdaMART is a hybrid of two techniques: LambdaRank and Multiple Additive Regression Trees (MART). It is an efficient and robust ranking algorithm mostly used for Information Retrieval (IR) tasks, in which the algorithm builds a model to rank web results under a user query. As applied to our scenario, the trained model will rank all tradable stocks at a given time. LambdaMART’s capability of handling missing values and its robustness to outliers make it the preferred learning algorithm for stock ranking and prediction.

MART is a gradient boosting algorithm that approximates the chosen target function additively by building one regression tree at a time [6]. The final learned model is the weighted sum of all regression trees. More specifically, given the user specified iterations number $M$, boosting builds $M$ regression trees sequentially to form the additive model

$$F(x) = \sum_{m=0}^{M} \beta_m f_m(x; y_m)$$

where $x$ is the input feature vector, $f_m(x; y_m)$ is the $m^{th}$ regression tree and $\beta_m$ is the learned weight for the $m^{th}$ tree. $y_m$ within $f_m(x; y_m)$ fully parameterizes the tree splits (choice of feature and threshold on the chosen feature). In each iteration, a new base learner $f_m(x)$ is built to minimize the overall loss function using gradient descent method. Under the assumption that $F_{m-1}(x)$ is smooth and differentiable near each training sample $x$, $y_m$ could be tuned, ideally to make $f_m(x)$’s output equivalent to $\frac{\partial L(F_{m-1}(x))}{\partial F_{m-1}(x)}$. So that the model performs the gradient descent in the steepest direction. However, under circumstances where no such parameters for $f_m(x)$ exist or it is infeasible to derive the solution due to computational cost, one could minimize the difference between the two. Thus, each base learner is formulated as

$$f_m(x) = -\rho_m h(x; a_m)$$

where $a_m$ are the parameters that defines each base learner $h(x; a_m)$, and they can be obtained by minimizing the squared difference between the gradient $g_m(x)$ and $h(x; a_m)$ over all training instances $x^t_i$.

$$a_m = \text{argmin}_{a_{\beta}} \sum [-g_m(x) - \beta h(x; a_m)]^2$$

In this way, MART reduces the complex cost function minimization problem to a stage-wise least-squares function minimization.

LambdaRank learns the probability that a given stock should be ranked higher than another by minimizing

$$C = \frac{1}{2} (1 - l_{ij}) \sigma o_{ij} + \log(1 + e^{-\sigma o_{ij}})$$

where $o_{ij}$ is the output of the $i^{th}$ ranking model.

$$l_{ij} = \begin{cases} 1 & \text{if } y_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

The cost function ensures that if the user ranks stock $i$ above stock $j$, then the output $(1 - l_{ij}) \cdot \sigma o_{ij}$ is minimized, which drives $o_{ij}$ larger (closer to 1). Conversely, if the user ranks stock $j$ above stock $i$, then the output is $l_{ij} \cdot \sigma o_{ij}$, which drives $o_{ij}$ smaller (closer to 0).
where $l_i$ and $o_{ij}$ are the differences between the labels and model outputs of two stocks respectively. For stock ranking as well as many IR tasks, ranking quality at head positions are typically more important than quality at body and tail, because one would naturally want to invest more on the predicted best performers. To accommodate this need of skewed learning focus, LambdaRank augments the gradient of the cost function above with Normalized Discounted Cumulative Gain (NDCG), which is a commonly used performance measure in IR, defined as

$$D_{CGT} = \sum_{i=1}^{T} \frac{2^i - 1}{\log(1 + i)}$$

$$NDCG_T = \frac{D_{CGT}}{\max(D_{CGT})}$$

where $T$ is the user specified truncation level, $l_i$ is the label for the $i^{th}$ document in the ranked list, $\max(D_{CGT})$ is DCG on the perfect ranking (ordered by $l_i$ descending). Typically, $l_i \in \{0,1,2,3,4\}$ with 4 meaning very relevant (profitable), 0 meaning not relevant. It is easy to observe that the more profitable stocks in the top T of the ranked result, the higher the DCG. The augmented gradient is defined as

$$\lambda_{ij} = \frac{\sigma}{1 + e^{\sigma o_{ij}}} |\Delta_{NDCG}|$$

where $\Delta_{NDCG}$ is the difference in NDCG by swapping the ranked positions of $Stock_i$ and $Stock_j$, $\sigma/(1 + e^{\sigma o_{ij}})$ is the gradient of the cost function with regard to the network outputs for $Stock_i$ and $Stock_j$.

With a well-defined gradient on ranking cost function from LambdaRank and a framework that breaks the complex function approximation problem down to stage-wise gradient approximation from MART, LambdaMART combines the merits from both algorithms for efficient and robust ranking optimization. Detailed algorithm pseudocode is listed in [4].

IV. RESEARCH DESIGN

A. Data

1) Order Flow and Transactions Data

We have gathered the order flow and transaction data from Shenzhen stock exchange from 03/01/2017 to 07/28/2017. Order data is the finest description of trading activities in the market. It contains all orders received by the electronic trading platform, with details about timestamp, price (if it is limit order), volume and order type. Market order, limit order and cancellations are all included and listed as different order types in the data. In general, market orders are more aggressive because they are matched immediately with the best opposite order in the limit order book until an opposite order can be paired with it. Since there are a variety of market order types in the ShenZhen stock exchange, some equivalent to limit orders, we adopt the notation of effective market/limit orders in [34] for brevity. Effect market order refers to orders that are filled immediately (partially or completely), while effective limit order refers to orders that wait in the limit order book for some time. Transaction data contains trading details about filled orders, with fields specifying the timestamp, execution price, execution volume, buy order details, sell order details, etc.

2) Aggregation and Feature Design

Orders and transactions arrive in irregular intervals. Since many price forecasters in the literature adopt the time-series model that aggregates features within fixed time spans, for a reasonable baseline and fair comparison, our ranker and price regressors are trained on aggregated features. Despite the information loss due to this aggregation, our results show that LambdaMART is capable of building profitable portfolios from such features. It is noteworthy that many continuous-time models had been built to couple with irregular spacing of high-frequency data [43], which is a future research direction with much potential.

Features we use for ranking and price modeling roughly fall into four categories: order features, limit order book features, technical indicator and news.

The order flow data contains precious information about the market participants’ trading strategy and their level of optimism. Previous research has shown that investors could be subject to waves of optimism and pessimism[1], suggesting that the sentiment on a stock could persist in the near future. A straight-forward measure of the market sentiment is the ratio of effective market orders to effective limit orders. We calculate the market order ratio on both bid and ask orders in regard to both order count and order value. Hewlett [44] observed that market orders tend to arrive in clusters, which could be explained by some well-known order execution strategies like batch ordering, with which large orders are executed in small blocks to minimize the impact on the market [45]. To capture the dynamics of order flows, we calculate the moving averages (MA) on the count and value of each order bucket and use the MA crossover signals as features to hint the beginning and end of an order cluster.

With complete order data, one can reconstruct the limit order book that accommodates all limit orders awaiting execution. Empirical studies have shown that the state of order book contains information about the future price movements. Ref. [34] showed that large price fluctuations could be attributed to gaps in the order book. Bid-ask spread was observed to be associated to securities’ return [46]. We use the gaps between each price level in the order book, as well as the bid-ask spread broken down to spread to bid and spread to ask as our order book features.

Practitioners use rules built on top of technical indicators to assist their trading. We adopt some crossover signals built on commonly used indicators such as Moving Averages (MA), Moving Average Convergence Divergence (MACD), Stochastic Oscillators (usually called KDJ indicators in Chinese markets) and Bollinger bands. It is noteworthy that some indicators, even basic ones like moving average2, are size when data is insufficient.
calculated differently in Chinese markets, and we find that features calculated in the market-specific way have higher association with stock ranking.

Financial news plays an important role in investors’ sentiment formation, which may drive money flow and stock prices. We implement a polite web crawler to retrieve financial news articles from five popular Chinese financial sites, namely, http://finance.qq.com/, http://finance.sina.com.cn, https://xueqiu.com/, http://www.caijing.com.cn/ and http://www.stockstar.com/. Retrieved news articles cover a wide range of news sources including official reports, general news about companies and market sectors, stock ratings and forum buzz. The crawled HTML webpages go through a pipeline of processors including text extraction, Chinese tokenization, deduplication and time stamping, stock and market sector classification. Text extraction is a much simpler task in Chinese webpages than in English ones because of the clear separation of Chinese text and HTML code written in English characters. A simple charset filtering suffices our needs. Chinese is written without spaces between words. Thus, in order to do text classification, we use a natural language processing package called ICTCLAS for word segmentation. Shingle hash deduplication is then performed on segmented N-grams to remove near duplicates from different sources. Related stocks of a given article are tagged by stock ticker matching (binary term frequency), and related market segments are identified with a naïve Bayesian classifier. With the contents of news articles classified and time stamped, we aggregate the count of news articles on each stock and its market segment at various time periods and use such features to give the learner a clue of dynamics in market attention and volatility.

B. Label

The rate of change in stock price is the basis of labels used for the ranker and regressors. At the data frequency we use, market dynamics can hardly be captured by just the stock price, which typically refers to the trading price of the last transaction that took place within a given time period. Due to gaps in the limit order book and sometimes large bid-ask spread, the closing prices at each second could flick up and down by a large amount, introducing non-trivial volatility to the return label. For example, shifting price within a spread of 5 cents on a 10-dollar stock could cause 0.5% fluctuation in the return label. For such calculated return can hardly be realized because the order that triggers the last price flip could be very small. To avoid such problem, we define our return as the ratio of average transaction price within \([t, t + 30 \text{ seconds}]\) to the current stock price. The calculated return for each stock at each second is used directly as labels for the neural network return regressor. For the ranker labels, all tradable stocks at each second are sorted on the calculated return, then grouped into 5 buckets with labels 4, 3, 2, 1, 0 respectively.

C. Model Parameter Tuning

We perform model parameter tuning with data from 03/01/2017 to 04/30/2017, which is further split into training (before 03/31/2017), validation (between 04/01/2017 and 04/15/2017) and testing (after 04/16/2017). The tuning is performed in a greedy fashion, i.e. find the best performing value for a parameter by randomizing it while keeping all other parameters constant and iterate until all parameters are chosen. For LambdaMART, the key parameters are number of leaves per tree and number of iterations (trees).

The number of leaves per tree specifies the complexity of each weak learner in boosting. The optimal choice for this model parameter depends on the shape of the target function to be approximated. It could be as low as two, which refers to a stump with two directed arcs connecting the same splitting node to two separate leaf nodes. Hastie et. al. [49] showed that boosting with stumps performs better than with deeper trees for solving the nested sphere problem, in which the target function is an additive quadratic multivariable equation with no interaction among each variable. Since it is unclear what the shape of our target function is, we varied the number of leaves from 2 to 256 and observed the following NDCG@3 on the testing dataset. It appears that increasing tree depth further than 30 does not benefit the overall model’s performance.

With number of leaves chosen, we set the number of trees to a relatively large value (5000), and apply early stopping to mitigate overfitting. Lodwich et. al. [50] compared several early stopping rules and suggested that Low Progress stopping criteria will more likely give better results given limited prior knowledge. Since LambdaMART seeks to maximize NDCG, we define the low progress rule as

\[
P_k(t) = 1000 \left( \frac{\max_{t-k+1}^t \text{NDCG}}{\text{avg}_{t-k+1}^t \text{NDCG}} - 1 \right) < \alpha
\]

where our choice of \(k\) and \(\alpha\) are 5 and 1, respectively.

For the neural network regressors, we need to decide the network structure as well as parameters related to alleviating local optima and overfitting, such as learning rate, momentum, weight decay, number of iterations and early stopping rule.

\[
\text{speed is low.}
\]
Since we have enough data for a separate validation set, learning rate, momentum and weight decay are set to small positive values, and we mostly rely on early stopping to stop training from a large number of iterations. Low progress early stopping criteria is defined in the same way as [51]. There is no general guideline for network structure design. Ref. [52] surveyed about a hundred papers that used neural networks for predicting stock price, return, risk and their composites. From their summarization, we observe that most designs use 1 to 2 hidden layers with no more than 60 nodes in each hidden layer. To find the structure that fits the problem we are trying to solve, we follow the observed guideline and experiment with 110 different network structures, including ten 1-hidden-layer networks with the number of hidden nodes spread from 10 to 100 at equal step size of 10, and one hundred 2-hidden-layer networks sampled in the same fashion. We observe that 2-hidden-layer networks with higher than 60 nodes in the first hidden layer perform better than 1-hidden-layer networks, and that 50 to 70 hidden nodes in the second hidden layer generally give better results than other choices. Our network structure is finalized as 61 input nodes, 100 nodes in the first hidden layer, 70 nodes in the second and 1 linear output node. Each layer is fully connected with the preceding one.

V. RESULTS

After tuning the parameters for both the LambdaMART ranker and the Neural Network (NN) regressor on parameter tuning dataset (from 2017-03-01 to 2017-04-30), the learning algorithms are trained on our training dataset (from 2017-05-01 to 2017-06-30), followed by the system evaluation performed on out-of-sample test dataset (from 2017-07-01 to 2017-07-28). Ranker is trained to optimize NDCG of volume weighted return at a 30 seconds lag. NN regressor is trained to predict the same return directly. To better demonstrate the modeling power of NN, we also train a network to predict the exact price at 1-second lag. These two NN models have the same structure and learning parameters, and they are denoted as return regressor and price regressor respectively. Features used for all three models are identical.

A. Metrics on Test Dataset

Ranker and regressor NDCGs are reported for ranking quality comparison. For the regressor, our evaluator ranks the stocks based on their predicted return, which can be viewed as investing in the most profitable stocks according to the learned model’s prediction. NDCGs for both ranker and regressor are averaged across 312,202 evaluations in the test dataset (22 trading days, 14191 seconds per day). For the return regressor, we also report the regressor errors in basis points (4.8546 L1 score means 0.00048546 mean absolute error in predicting the return on testing dataset), averaged over 546.4 million evaluations (about 1750 tradable stocks per second per day). The reported error in price regressor is CNY. Note that the minimal tick for all stocks in the target market is 0.01 CNY, and an L1 score of 0.004 gets the regressor very close to the actual price. NDCG is not applicable to the price regressor because there is not much sense in ranking stocks on their prices.

As shown in the table, the ranker outperforms the return regressor drastically in terms of NDCG. This significant ranking quality difference is also confirmed by simulated trading profit shown in the next section, suggesting that although the NN learners are capable of approximating the stock price and return closely, the lack of ranking concept in the objective function makes the learner less reliable for constructing portfolios directly. More specifically, in LambdaMART, an error at the head of the return distribution is penalized more than an error of the same scale at the body and tail, but regressor models would treat such errors equally. The ranker may perform worse than regressors at the body and tail, but the ranking in the head is what matters the most in investment.

B. Trading Simulation

To demonstrate the profitability of our trained system, we set up some basic trading rules to calculate the simulated profit on the testing dataset. Since short positions are not supported in our target market, our trader only takes long positions. Our trader starts by dividing the available fund into 3 buckets. For each unallocated bucket, it opens a long position on the stock with optimal ranking. Since there are 3 buckets, we could long the top 3 stocks at once, corresponding to NDCG@3 reported in the previous section. Buying orders are executed at optimal ask price in the limit order book. We assume there is enough ask quotes at the optimal ask price to use up all fund in the bucket, which is unrealistic for big funds. But since designing an order execution strategy is out of the scope of this research, we make this assumption on both the ranker and regressor for brevity. We close the position at the optimal bid price if the stock falls out of top 100 and there is no trading halt [53]. If a halt is imposed, the position will be closed immediately after the halt is lifted.

### Table I: Metrics on Out-of-Sample Testing Dataset

<table>
<thead>
<tr>
<th>Metric</th>
<th>Ranker</th>
<th>Return Regressor</th>
<th>Price Regressor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@1</td>
<td>87.8221</td>
<td>22.1763</td>
<td>-</td>
</tr>
<tr>
<td>NDCG@2</td>
<td>85.0587</td>
<td>18.3154</td>
<td>-</td>
</tr>
<tr>
<td>NDCG@3</td>
<td>82.7252</td>
<td>10.9981</td>
<td>-</td>
</tr>
<tr>
<td>L1</td>
<td>-</td>
<td>4.8546</td>
<td>0.0040</td>
</tr>
<tr>
<td>L2</td>
<td>-</td>
<td>104.8171</td>
<td>0.0002</td>
</tr>
<tr>
<td>RMS</td>
<td>-</td>
<td>10.2380</td>
<td>0.0165</td>
</tr>
</tbody>
</table>

Transaction costs are deducted from the profit of each position. Stamp tax (0.1%), government fees (0.02%) and broker commissions (varies per broker from 0.00% to 0.1%) mount up to an estimated total of 0.23% on the closing value of the position. Table III shows the statistics about model
profits. Investment decisions made by the ranker are much better than those made by neural networks in terms of both absolute return and risk-adjusted return. Note that the average return per position from the neural networks model is not sufficient to cover transaction cost and the asset value approaches zero with the max drawdown over 1.

Fig. 2 shows a one-day cumulative gain of LambdaMART and Neural Networks, compared to the market index on simulated trading. The ranker portfolio performs better than the overall market index, while neural network portfolio suffers from transaction costs and the asset value keeps regressing after a brief hike in the morning.

To further understand the reason why low regressor test error does not lead to good profitability in the NN model, we plot the outputs of the ranker, return regressor, price regressor, as well as the actual price of a stock in the following figure. We attribute the poor profitability of the regressors to the lag in predicted prices from index 174 to 200, the price regressor forecasts the price movement in the wrong direction about 80% of the time, and the return regressor generates unprofitable buy signals for three times, each with a 2 seconds lag. Although the ranker output in the sideways period is volatile as well, it never pushes this stock to the top 3 positions.

VI. CONCLUSION AND FUTURE WORK

In this research, we design a stock ranking system that uses LambdaMART to predict the best-performing stocks in an intraday scale with data from ShenZhen stock exchange. We demonstrate that the ranking algorithm has significantly better performance in portfolio selection than neural network regressors in terms of ranking quality and simulated profits. Both the ranker and the regressor are trained on features extracted from diverse trading practices in the hope that these features, collectively, could reflect a majority of factors involved in traders’ decision making. Trading simulation is performed on out-of-sample test data under rigorous conditions with respect to trading price and transaction cost, and the results demonstrate strong profitability of the ranking algorithm.

Aligning terabytes of order and transaction data with online news for forecasting is an exciting area that is barely explored. We expect to continue our work in the following areas. First, although LambdaMART is quite efficient and robust, it is not straightforward to recognize price patterns which is an important factor in many traders’ decision-making process. The pattern representation transformation widely used in deep learning is a good supplement to this shortage of LambdaMART. It can either be used as a pre-processing model that generates features to be consumed by the ranker or used as the base learner in the tree ensemble. Recent developments in high frequency financial econometric shed light on more features we could use for better ranking result [54]. And finally, order execution can be formulated into a constrained multi-objective optimization problem for which learning models would seek to optimize the execution price and certainty within the constraints of time and rapidly changing market microstructure.

REFERENCES
