ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Σχολή Τεχνολογιών Πληροφορικής και Επικοινωνιών

Τμήμα Ψηφιακών Συστημάτων

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Πληροφοριακά Συστήματα & Υπηρεσίες

Κατεύθυνση :

Μεγάλα Δεδομένα και Αναλυτική

Online Learning for Market Making: An Empirical Evaluation

Κωνσταντίνος Πρωτόπαπας

Α.Μ. ΜΕ2128

Scope of Work

The primary function of a Market Maker in the stock exchange is to enhance liquidity by actively placing buy and sell orders for securities, while simultaneously seeking to capitalize on fluctuations in stock prices. In contemporary stock exchanges, most of the trading takes place through computerized systems, thereby necessitating a heightened emphasis on the rapidity and intricacy of transactions. This is accomplished with the broad usage of high-end software and algorithms deployed by Banks, Hedge Funds, Proprietary Trading firms etc.

The scope of this thesis is the development, application and comparative study of Online Training methodologies in the field of Market Making applications. To accommodate the inherent variety of approaches to this problem, we begin with a review of relevant parts of the financial field, to identify parameters and limitations which will prove significant later.

We move on with a review and categorization of algorithmic approaches to identify representative candidate methodologies to compare against. We make an extended presentation of various meta-algorithms, especially Expert and Bandit problem variants which are useful for challenging data, especially those exhibiting Concept Drift.

In the latter chapters of this work, we present a novel approach for Online Training and lay the groundwork for reproducible, comprehensive application, as well as document and discuss the results.

Lastly, we perform comparative research between methodologies, documents and discuss the results in order to derive insights furthering the field.

Contents

1.Introduction

1.1 The Market and the Market Maker

The stock market functions as a centralized platform where investors—both individuals and institutions—buy and sell equity securities, more commonly known as stocks or company shares. Prominent exchanges around the world, including the New York Stock Exchange (NYSE), Nasdaq, the London Stock Exchange (LSE), and the Frankfurt Stock Exchange, handle millions of transactions daily, reflecting the market's vast scale and liquidity. [40].

These exchanges are monitored by regulators to make sure trading stays fair, honest, and transparent. While many types of financial products are traded there, this thesis focuses only on stocks. A stock gives the holder a piece of a company's assets and profits—basically, it means owning a small part of that company. [21].

At the core of a stock exchange's operation is a matching mechanism between buyers and sellers. Participants enter the market with the intention of buying or selling shares, and transaction prices are determined by market dynamics—specifically, the balance between supply and demand at any given time. Orders submitted by market participants typically fall into two categories: market orders and limit orders [31].

- Market orders are executed immediately at the best available price. When a participant initiates a market purchase or sale, the order is fulfilled based on prevailing market conditions, without price specification [31].

- Limit orders differ in that they include a specified price at which the trader is willing to buy or sell. A limit buy order sets a maximum price to purchase, while a limit sell order defines a minimum price to sell. These orders remain active in the system until they are either matched with an appropriate counterparty or manually canceled. Since their execution depends on market movements, they may not always be fulfilled. [28].

Trades occur when opposing orders—one to buy and one to sell—agree on both price and quantity. This agreement is referred to as a match. Given the large number of simultaneous orders in modern markets, stock exchanges implement a systematic execution policy. Priority is typically granted to orders priced most favorably relative to the current market rate—higher buy

bids and lower sell asks. When prices are equal, the order placed earlier takes precedence (time priority) [9].

A stock exchange functions similarly to other marketplaces, governed by its own rules and institutional frameworks. Its core objectives include the promotion of fair trading, accurate price formation, and adequate market liquidity [28].

Participants in equity markets can fulfill different roles. Investors tend to hold stocks for extended durations, seeking long-term appreciation, whereas traders frequently engage in buying and selling over short intervals to profit from minor price shifts. A distinct participant class is the Market Maker (MM), whose presence is critical to maintaining liquidity [44].

A Market Maker is a specialized trader or firm that continuously quotes both bid and ask prices for specific securities. Their profitability depends on the spread of the bid-ask prices, the difference between the highest price they are willing to pay (bid price) and the lowest price at which they are willing to sell (ask price) [21]. By consistently providing these quotes, Market Makers help reduce transaction delays and mitigate liquidity risk. However, they also face the challenge of price volatility, as securities held in inventory may depreciate before being resold [44].

Market Makers manage their operations using a tool known as the Limit Order Book (LOB), which tracks all outstanding limit orders for a given stock. When a limit order is placed, it is recorded in the LOB and awaits execution. Buy orders are filled only when a seller agrees to the specified or a lower price, while sell orders are executed only if a buyer is willing to meet or exceed the set price. The LOB thus enables orderly and rule-based trade matching [8].

1.2 The Emergence of High Frequency Trading (HFT)

HFT is mainly defined through its characteristics, rather than a top-down definition based on theoretical principles. In this spirit, key characteristics associated with HFT include:

- **High Speed and Low Latency:** This is a defining characteristic. HFT trades are commonly concluded within a span of milliseconds or less.
- **High Order-to-Trade Ratio:** HFT strategies often involve submitting a large number of orders that are frequently canceled or modified within very short time frames. The number of orders sent can be significantly higher than the number of executed trades.
- **Short Holding Periods:** The aim of HFT is to profit from small price movements and market inefficiencies within a span of minutes or even seconds.

- **High Turnover Rates:** Due to the short holding periods and high frequency of trading, HFT firms have very high turnover rates in their portfolios.
- **Use of Sophisticated Algorithms:** High-Frequency Trading (HFT) relies on sophisticated algorithms capable of processing large volumes of market data, spotting trading opportunities, and executing orders automatically according to predefined rules.
- **Co-location and Direct Market Access:** HFT firms often utilize co-location services offered by exchanges to minimize latency. They also rely on direct market access (DMA) to bypass intermediaries and send orders directly to the exchange.
- **Proprietary Trading:** HFT firms typically trade for their own account (proprietary trading) rather than on behalf of clients.
- **Focus on Liquidity and Small Profits:** Many HFT strategies aim to profit from providing liquidity by placing bid and ask orders (market making) and capturing the bid-ask spread. They often seek to make small profits on a large volume of trades.

Having listed these characteristics, we include the definition of HFT used by EU regulatory authorities [20]

(40) *'high-frequency algorithmic trading technique'* means an algorithmic trading technique characterized by:

- (a) infrastructure intended to minimize network and other types of latencies, including at least one of the following facilities for algorithmic order entry: co-location, proximity hosting or high-speed direct electronic access;

- (b) system-determination of order initiation, generation, routing or execution without human intervention for individual trades or orders; and

- (c) high messaging intraday rates which constitute orders, quotes or cancellations;

and the definition used by Nasdaq [42]

High Frequency Trading (HFT)

High-frequency trading (HFT), also known as algorithmic or algo trading, involves the use of computer-driven algorithms to carry out trades. There are two main forms. The

first, known as execution trading, uses algorithms to carry out a predetermined order—often a large one—by breaking it into smaller parts and executing over time to achieve the best price. The second type doesn't follow a preset order but instead scans the market for small, short-term opportunities to profit. It's estimated that algorithmic trading now accounts for about half of the total stock trading volume in the U.S.

It becomes apparent that HFT requires a financial infrastructure, a technical infrastructure, a regulatory infrastructure and a methodological infrastructure, before the efficiency of different algorithms can be discussed.

1.3 Finance-Centric Background Information

1.3.1 Limit Order Book (LOB)

A significant part of the methodological infrastructure is the concept of LOBs, which provide a scalable framework to facilitate trade by matching allowing every trader to post buy or sell orders [26]. We highlight this framework because its mathematical formalization allows the application of algorithmic tools automating the matching between orders, thus paving the crossroads between the financial and technical frameworks required by HFT.

A formalized definition [25] of the most common form of LOBs is as follows:

An order $x = (p_x;\ \omega_x;\ t)$ placed at time t_x with price p_x and size $\omega_x > 0$ (respectively, $\omega_x < 0$) is a commitment to sell (respectively, buy) up to $|\omega_x|$ units of the traded asset at a price no less than (respectively, no greater than) p_x.

We also include two more supplementary definitions which are of interest in the computational side of LOB use:

The lot size of an LOB is the smallest amount of the asset that can be traded within it. All orders must arrive with a size $\omega_x \in \{\pm k\sigma \mid k = 1,2,\dots\}$.

The tick size π of an LOB is the smallest permissible price interval between different orders within it. All orders must arrive with a price that is specified to the accuracy of π.

The fundamental part of the LOB innovation is that an order remains in the queue until it is matched by an order of the opposite type (buy-sell) or until it is cancelled [26]. A constant influx of orders will ensure that, over time, both buy and sell orders are executed, and the incorporation of electronic LOB platforms guarantees that a worldwide audience can express their interest by placing such orders. Formally [25]:

Let $x = (p_x; \omega_x; t)$ be a buy (or sell, respectively) order arriving immediately after time t, $b(t) = max_{x \in B(t)} p_x$ the bid price and $a(t) = min_{x \in A(t)} p_x$ the ask price.

Then:

- If $p_x \leq b(t)$ (respectively $p_x \geq a(t)$, then x is a limit order that becomes active upon arrival and does not cause changes in $b(t), a(t)$

- If $b(t) < p_x < a(t)$, then x is a limit order that becomes active upon arrival, causing an increase in $b(t)$ (respectively, a decrease in $a(t)$) tp p_x at time t_x

- If $p_x \geq a(t)$ (respectively, $p_x \leq b(t)$), then x is a market order that immediately matches one or more active sell (respectively, buy) orders upon arrival. The new value of $a(t)$ is:

$a(t_x) = min(p_x, q)$, where $q = arg\, min_{k'} \sum_{k=a(t)} \quad n^a(k,t) > |\omega_x|$
and respectively,

$b(t_x) = max(p_x, q)$, where $q = arg\, max_{k'} \sum_{k=k'}^{b(t)} \quad |n^b(k,t)| > \omega$

1.3.2 The Economic Impact of Price Prediction in LOBs

Essentially, an incoming order p_x is matched by a prioritized vector of the opposite type. If its execution leaves a remainder, the next prioritized order of the opposite type is considered until there are no further active orders with a price which makes them eligible. It can be seen that this structure is essentially optimized for automation in a computational context. This means that the open question for the financial sector is how to automate the selection of the p_x, ω_x, t parameters that make up an order. Even limited insight into an optimized price, quantity and correct time to place an order may influence the chances of a limit order being executed, which means direct financial gains (or limitation of losses in the worst-case scenario).

Let us expand on the important distinction between limit orders and market orders, which was hinted in the above equations. The active buy and sell orders at any given moment t define a price range $[b(t), a(t))]$ (the case of $b(t) < a(t)$ is possible but not expected under normal circumstances). A "market order" is certain to be carried out (at least in part), taking place within this price range. A limit order is an order that will be carried out at a pre-defined price and may or may not be inside this range. Thus, a limit order may remain stagnant [26] until an eligible, opposite order is placed.

In a laboratory setting, this would not be an issue, as infinite time could be devoted to an experiment. In a real-world setting, specific volumes of stocks are expected to be bought or sold at opportune times. Therefore, a stagnant limit order represents an amount of stock which can miss the window of opportunity within which it would produce a certain amount of net profit. This shows the importance of being able to predict future market trends, so that limit orders can be placed with a reasonable expectation of a match within a set time frame.

This need brings us to the main focus of this work, the application of predictive algorithms in HFT. Within the scope of this work, the main use of LOBs is that they provide a protocol and a context for the development of algorithms which can automate the submission, matching and execution of orders. This is a crucial requirement for all definitions of HFT, as the sheer volume of transactions required each second is beyond any human capability to carry out and monitor. Reliable predictive algorithms reduce the need for manual human oversight and provide fine-tuning of all the parameters we have mentioned in order to make matching of buy/sell orders more efficient.

1.3.3 LOBs in the Context of Algorithmic Development

Monitoring is at the center of the development of algorithms used in HFT Market Making. Firstly, the existence of LOBs provides a scalable framework for tracking buy and sell orders, producing very granular datasets. Secondly, LOBs provide a framework to track gains and losses in a standardized manner, answering one of the fundamental problems in machine learning - having access to clear answers about the gains or losses of a particular decision. Thirdly, HFT provides researchers with high volumes of very granular data, which in turn addresses the need of machine learning algorithms for well-labeled, large data sets (the latter being a common limitation in training neural networks).

Based on that, a key question when applying machine learning to Limit Order Books is whether the current LOB state contains useful information for predicting future price changes. [26]. This question has been approached in many different ways, some of which will be presented in the following chapters. This question is of great interest to Market Making firms, which manage large volumes of stock trading.

First, being able to moderately predict even the immediate movements of the market allows a Market Maker to adjust their strategy [25] [5] between aggression (prioritizing timely execution of bids and incurring costs) and passivity (aiming for increased gains but potentially decreasing the chance for an order to find a suitable match). Secondly, being able to predict the market allows a Market Maker to estimate the price impact their own moves may have on the market [25]. Unplanned consequences may result in market volatility, so this capability provides both an individual and a communal gain.

Lastly, we need to stress a specific quality of the real-world use of LOBs versus their academic interest. In day-to-day trading, past states of the LOB are known, but the new data are being generated in real-time. This property of the field means that we need algorithms that do not require a ready, finalized dataset to train and implement, for two reasons:

- Firstly, because the interest of Market Makers is concentrated on new data and the ability to predict the next few states. This means that not all data are treated equally, both because the data form a time series and because the latest data may be more informative to prediction than older LOB states.

- Secondly, because there is no guarantee that the market distribution will remain constant for any length of time. For any number of reasons, shifts are expected to occur, thus a model will require updating to catch up, a phenomenon described as

Concept Drift. In contrast, a historic dataset used for research purposes will always have a set distribution which can be known in theory, even if it is complex.

These limitations will inform the selection of methodologies which will be described in two ways. First, we focus on online learning methodologies, as they are explicitly developed to deal with incoming data being generated in real-time. Secondly, the mitigation of Concept Drift means that we will have to make special mentions to methodologies which can adapt to changing data distributions.

1.3.4 Market Making

So far, we have described detailed models and protocols, as well as very broadly defined the HFT sector. Here, we will focus on Market Making and the set of entities that bring these two sides together.

The theoretical underpinnings of Market Making rest on a small number of basic assumptions [25], namely:

- A dealer who faces a stochastic arrival of buy and sell orders
- Each trade incurs a cost
- The dealer is risk-averse and faces costs related to holding inventory
- The true value of the asset follows a known distribution

The work of Ho, T., & Stoll, H. R. [29] places emphasis on the impact of inventory management, which may influence prices as the dealer chooses to increase or decrease their inventory. A different, important dimension of Market Making which concerns this work was put forth by Glosten, L. R., & Milgrom, P. R. and concerns information.

In this model, there is a sharp distinction between uninformed investors, who have access to public information (past transaction prices, bid & ask and other publicly available information) and informed investors who also have access to some non-public information [24]. This distinction is not just qualitative, as the bid-ask spread is widened in order to account for expected losses to informed investors. At this level, the importance of predictive models is seen from the perspective of the Market Maker and the perspective of an investor, as a successful prediction made by an

uninformed investor may make up for a part of the information divided between themselves and an informed investor.

Moving on the work of Grossman, S. J., & Miller, M. H. [27], they do not focus on the individual Market Maker (as the two previous works did), but rather on the field of Market Making as a whole. Their work begins by regarding Markets Makers as intermediaries of the market and attempts to provide a theoretical framework on how they can influence market liquidity. This work, too, places emphasis on the timely execution of orders, as an investor may wish to make transactions within a specific timeframe when it is opportune to do so, rather than wait. The Market Makers face a cost to maintain a presence in the market (modeled as fixed costs in their work) and are in competition with each other.

Based on these principles, the authors show that market liquidity will be determined by the balance between the costs incurred by the Market Makers and the demand of investors for immediacy. In other words, increased fixed costs under this model will translate to fewer market makers and lower liquidity. Conversely, an increase in the demand for immediacy translates to a market able to support a larger number of Market Makers and higher liquidity. The demand for immediacy is tied to the existence of HFT, but now that HFT is an established, worldwide domain this part of the equation is not really flexible. On the other hand, the operating costs of the Market Maker can be greatly influenced by the technological and mathematical advances in their disposal.

1.4 Computational-Centric Background Information

1.4.1. Online Learning Algorithms

In most branches of machine learning algorithms, training supposes a pre-existing, curated data set. Online learning is a subcategory of algorithms developed to account for applications in which the training data arrive in discrete packets and thus prediction and training must take place concurrently with what is essentially partial training data. This is the reasoning behind the claim that Online Learning algorithms match very well with the complex and fast paced HFT environments discussed priorly. The repeated and swift processing of new, real-time data over extremely short time frames, sometimes even in the micro or mini seconds.

In this approach, training and prediction are iterated at every step ($t = (1,2,3,\dots,T)$) of execution [41].

- At each step, the algorithm receives a set of data $x_t \in X$ and outputs a prediction vector $\hat{}_t \in Y$.
- After the true value$\hat{}_t$ is received, the Loss Function $L(\hat{}_t)$ is calculated.

- The goal is to minimize the cumulative loss $\sum_{t=1}$ $L(\hat{t}, t)$ over a horizon T.
- Loss:

  $L(\hat{t}, t) = \uparrow_t - \hat{}_t|$

  Regret:

  $R_T = \sum_{t=1}^{T}$ $L(\hat{t}, t) - min \sum_{t=1} \bar{L}(\hat{t}, t)$

Algorithm Categorization

Given the numerous constraints described so far, there are two ways relevant to this work to categorize established Online Learning algorithms. The first is the top-down perspective of categorization according to the core learning algorithm, so that approaches stemming from similar philosophies or using similar analytical tools are grouped together.

The second approach, however, is more suited for the present work, which concerns application to HFT. The main feature of Online Learning algorithms is that training is performed in an iterative manner using incoming batches of data, rather than a full dataset. The result is that the underlying distribution, statistical characteristics and even qualitative characteristics may change over time (Concept Drift). This categorization allows the user to select approaches according to the volatilities and variety of underlying structures these methods are expected to contend with.

1.4.2. Categorization According to the Core Algorithm

- **Online Gradient Descent (OGD)**
  - Stochastic Gradient Descent [52] [30]: The most fundamental online learning algorithm. Model parameters are updated based on the gradient computed using a single data point (or a mini batch). It's computationally efficient and can adapt quickly to new data.
  - Momentum Methods [17]: In these approaches, learning is accelerated using the exponentially decaying average of past gradients, helping to navigate ravines and speed up convergence. Online versions adapt this concept.
  - Adaptive Learning Rate Methods (e.g., AdaGrad [19] [39], RMSprop [51], Adam [33]): The learning rate is generally adjusted for each parameter based on the historical gradients. Online versions of these approaches maintain per-parameter learning rates that adapt as new data arrives.

- **Online Linear Models**
    - Perceptron [34] [16]: An approach for binary classification. Weights are updated based on misclassified examples.
    - Passive-Aggressive Algorithms (PAs) [16]: A family of algorithms used for classification and regression. They aim to make minimal changes to the model while still correcting misclassifications or errors aggressively when necessary. Variants include PA-I and PA-II, which differ in their updating rules and aggressiveness.
    - Online Logistic Regression: An extension of logistic regression to the Online Learning domain [16], typically using online gradient descent or its variants for parameter updates.
    - Follow-the-Leader (FTL) and Follow-the-Regularized-Leader (FTRL) [12]: These are essentially meta-algorithms providing a framework for online learning. FTL simply chooses the model that has performed best so far. FTRL adds a regularization term to the objective function at each step, promoting stability and better generalization. Algorithms like online logistic regression and linear regression can be implemented using the FTRL framework.

- **Online Support Vector Machines (SVMs)**
    - Algorithms like Pegasos (Primal Estimated sub-GrAdient SOlver for SVM) [46] are designed for online large-scale linear SVM training. A stochastic sub-gradient descent approach is used on the primal form of the SVM objective.

- **Online Tree-Based Methods**
    - Hoeffding Trees (VFDT - Very Fast Decision Tree) [18]: Designed for mining evolving data streams, these algorithms efficiently adapt their structure to the influx of new data, using Hoeffding bounds to decide when to split nodes. Variants like CVFDT (Concept-adapting Very Fast Decision Tree) can handle concept drift.
    - Online Random Forests [36]: Ensembles of Hoeffding Trees that can provide more robust and accurate predictions in online settings. New trees can be added or existing trees updated as more data becomes available.

- **Online Neural Networks** [45]

- Neural networks can be trained online using backpropagation with stochastic gradient descent or its variants on individual data points or mini batches. This is common in applications with continuous data streams.

1.4.3. Categorization According to the Handling of Concept Drift

- **Passive Methods**
  - This approach continuously updates the model using new data, hoping to adapt gradually to changes. Most basic online learning algorithms (e.g. SGD) fall into this category. Their adaptation speed depends on the learning rate.

- **Drift Detection Methods:** These methods explicitly monitor incoming data or model performance to detect significant changes in the data distribution. If concept drift is detected, they respond with a pre-set adaptation mechanism. Examples include:
  - Statistical Process Control-based methods (e.g., DDM - Drift Detection Method [22], EDDM - Early Drift Detection Method [6]): Error rates or other performance metrics are subjected to statistical tests in order to detect significant deviations.
  - Window-based methods (e.g., ADWIN - Adaptive Windowing [7]): This approach maintains a sliding window of recent data and compares statistical properties of different sub-windows in order to spot changes. When concept drift is detected, older data might be discarded.

- **Active Adaptation Methods:** If concept drift is detected, these methods employ specific strategies to adapt the model. Sample strategies [23] [38] [10] include:
  - Retraining: Partially or fully retrain the model on the most recent data.
  - Model Updating: Adjust the model parameters more aggressively than passive methods or change the model structure outright.
  - Ensemble Methods with Drift Handling: Maintain an ensemble of models, where older models might be down weighted or replaced as drift occurs. New models can be trained on recent data.

- Adding/Removing Model Components: In the case of more complex models like neural networks or decision trees, drift might necessitate adding new neurons/branches and/or pruning existing ones.

2.Use of Online Learning algorithms in Market Making

2.1 Summarizing Demands and Limitations

The use of Online Learning in Market Making can be easily adapted to make use of LOBs. The main advantage of LOBs in this context is that they provide a clear, sequential framework along with an easily identifiable result for a Loss Function (comparative financial performance). M. Kearns and Y. Nevmyvakaz [32] provide examples of leveraging reinforcement learning in HFT in order to prioritize the aggressiveness of placing orders in a LOB to meet set goals efficiently. Their example illustrates the constraints arising from the need to meet a set total buy volume within a set timeframe while minimizing the cost of this acquisition and discuss how these constraints drive the application of different strategies. The main takeaway from their work is that the power of this approach lies in the automatic, responsive fine-tuning of model parameters, rather than discovering novel, unintuitive approaches.

The work of T. Spooner et al. [49] showcases implementations of reinforcement learning in a data-driven simulation of a LOB. A key takeaway from this work is the comparison between different versions of a reward function, showing that a simple “profit and loss” approach can be too responsive to market trends and can be outperformed by functions introducing what is essentially a form of inertia.

A later work of T. Spooner and R. Savani [48] moves the field forward by implementing an adversarial reinforcement learning scheme in Market Making. In this approach, an agent competes against an adversary in order to drive model improvements. This specific work models a zero-sum game between the agent (the investor’s alter ego) and the adversary (which models the variety of other investors, in essence “the market”). The simulation reaches a Nash equilibrium, indicating the stability of this solution.

Conceptually, the next step is best described by the work of J. Abernethy and S. Kale [1], showcasing the use of meta-algorithms to find an optimal solution in Market Making. In particular, once the assumptions and strengths of various strategies are outlined, each strategy is regarded as an “expert,” and a regret-minimization algorithm is employed under the learning-from-experts framework. Their result of applying this approach on available market data illustrates that there is a need for adaptive algorithms, as different meta-algorithms may outperform each other in different circumstances.

The drive to derive more and more elaborate models generated a desire to seek model-free solutions to machine learning applications in Market Making. Such an approach was studied by Y. Nevmyvaka and M. J. Kearns, using a Q-learning methodology for choosing optimal strategies in Market making. Moving onward, the work of P. Kumar [43] explores one such path by employing a Deep Recurrent Q-Network approach in order to eliminate the need for manual feature design. A key takeaway from this body of work is that this approach seems viable and may outperform other models, but hinges on the availability of vast amounts of training data to properly train the agent.

2.2 Ensemble Methods

Given the progress and limitations described above, for the rest of this work we will focus on ensemble methods for Market Making. Their being essentially meta-algorithms, where "experts" correspond to different models allows for adaptation in situations where Concept Drift would make one or more models underperform by bringing forth (assigning more weight) to models that fit the current data better.

We can formulate the core logic using a simple example [11], where there is always an expert who cannot make a mistake under any circumstances:

The aim is to predict an unknown sequence $\hat{}_1\hat{}_2\hat{}_3,\ldots$ of bits $\hat{}_t \in \{0,1\}$. Each time the forecaster makes a guess ($p \in \{0,1\}$) for $\hat{}_t$, based on the predictions of N experts. In this example, the advice is a binary vector $(f_{1,t}, f_{2,t}, \ldots, f_{N,t})$, where $f_{i,t}$ corresponds to the to the prediction made by the expert with index i for the bit $\hat{}_t$.

Having assigned an initial weight w_i to the experts, the prediction is made according to a set of rules which take into account the number of experts pointing to a certain prediction and their weights. In this example, $w_i \in \{0,1\}$ and a prediction $p_t = 1$ is made if and only if among the experts with $w_i = 1$, the majority makes a prediction $f_{i,t} = 1$. In other words, our simple rule is a majority rule where only experts with $w_i = 1$ are allowed a vote.

Next, we compare the prediction of each expert to the true value of $\hat{}_t$ and for every prediction $f_{k,t} \neq \hat{}_t$ we set $w_k = 0$, thus eliminating every expert who made a wrong prediction from future voting.

2.2.1 Expert Models

In real-world examples, we need to account for the fact that all experts are fallible and that our goal is the optimization of our overall prediction process. We can then formulate the prediction with expert advice as follows [211]:

Parameters:

- D: The set of all possible decisions the learner can make.
- Y: The set of possible outcomes or observations from the environment.
- l: A loss function that measures how costly a decision is given the actual outcome
- E: The collection of indices representing the available experts.

For each round $t = 1, 2, \ldots$

1. The environment chooses the next outcome $\hat{}_t$ and the expert advice $\{f_{E,t} \in D\}$; the expert advice is revealed to the forecaster
2. The forecaster chooses the prediction $p_t \in D$
3. The environment reveals the outcome $\hat{}_t \in D$
4. The forecaster incurs loss $l(p_t, \hat{}_t)$ and each expert in E incurs loss $l(f_{E,t}, \hat{}_t)$

Throughout this process, the goal is to minimize the cumulative regret of each expert in E [11] [13]:

$$R_{E,n} = \sum_{n}^{n} \quad (l(p_t, \hat{}_t) - l(f_{E,t}, \hat{}_t)) = L_n - L_{E,n\,t-1}$$

Where

- $L_n = \sum_{t-1} \quad l(p_t, \hat{}_t)$ is the forecaster's cumulative loss
- $L_{E,n} = \sum_{t-1}^{n} \quad l(f_{E,t}, \hat{}_t)$ is the cumulative loss of expert E

and for the forecaster to make predictions so that the regret is minimized for all sequences of outcomes.

We also introduce the notion of instantaneous regret at time t, measured relative to expert E, defined as:

$$r_{E,t} = l(p_t, \hat{t}) - l(f_{E,t}, \hat{t})$$

We now see that expert models can be refined at three main levels:

- The nature of the set of experts
- The weighting mechanism is used to leverage experts so that the forecaster makes a prediction
- The loss functions for the forecaster and experts (which in the latter case may feed into updating expert weights)

The latter two approaches to developing different strategies will be shown through the examples below.

2.2.2 Weighted Average Prediction

The weighted average prediction framework [11] [14] makes a prediction according to

$$p_t = \frac{\sum_{t-1}^{n} w_{i,t-1} f_{i,t}}{\sum^{n} w}, \quad {}_{j-1} \quad {}_{j,t-1}$$

thus, assigning a normalized weight to the prediction vector using the weights calculated over all previous predictions. The natural feedback loop in this approach is to define an arbitrary function to re-calculate weights at each step, which will be decreasing as the cumulative loss $L_{i,t-1}$ increases, in order to minimize overall cumulative loss.

2.2.3 Multiplicative Weights Algorithm

The Multiplicative Weights algorithm begins by supposing that the first action is chosen at random. Afterwards, the chance of our choice is re-calculated based on the feedback returned by the forecaster. In doing so, the probability distribution for our choices is continually updated, reflecting the costs of different choices and highlighting the most successful models. Formally:

Parameters:

- $\eta \leq \frac{1}{2}$: fixed parameter
- I: action space

Initialization:

- $w_i = 1$ for all actions $i \in I$

Then, for each round $t = 1,2,\ldots$

1. Choose an action I_t based on the probability distribution w_i (normalized so that $\sum^i \quad w_i = 1$)
2. The costs of the decision, cost, are revealed
3. Penalize the decisions by re-calculating their weight based on the cost associated with each possible decision

   $w_{t+1,i} = w_{t,i}(1 - \eta\ cost_{i,t})$, for all $i \in I$

2.2.4 Polynomially Weighted Average Forecaster

Another approach to weighing [11] is to define the prediction as

$$p_t = \frac{\sum_{t-1}^{N} \quad (\sum_{s-1}^{t-1} \quad (l(p\hat{s},s) - l(f_{i,\hat{s},s})))_{+}^{p-1} f_{i,t}}{\sum_{j-1}^{N} \quad (\sum_{s-1}^{t-1} \quad (l(p\hat{s},s) - l(f_{j,\hat{s},s})))_{+}^{p-1}},$$

where the "+" notation denotes the positive terms.

It can be shown [11] than that the regret of the polynomially weighted average forecaster satisfies

$$L_n - min_{=1,2,..,N} L_{i,n} \leq \sqrt{n(p-1)N^{p}}^{\frac{2}{p}}$$

For all $p \geq 2$, the per-round regret converges to zero at a rate $O(n^{-\frac{1}{2}})$ uniformly over the outcome sequence and expert advice. The choice $p = 2$ provides a simplified algorithm, whereas the choice $p = 2\ ln\ N$ for $N > 2$ approximately minimises the upper bound, leading to

$$L_n - min_{i=1,2,..,N} L_{i,n} \leq \sqrt{ne(2\ ln\ N - 1)},$$

yielding a significantly better dependence on the number of experts N.

2.2.5 Exponentially Weighted Average Forecaster

The weights distributed to the experts are of the form

$$w_{i,t-1} = \frac{e^{\eta R_{i,t-1}}}{\sum_{j=1}^{N} e^{\eta R_{i,t-1}}}$$

and the weighted average forecaster simplifies to

$$p_t = \frac{\sum_{i=1}^{N} e^{\eta(L_{t-1} - L_{i,t-1})} f_{i,t}}{\sum_{j=1}^{N} e^{\eta(L_{t-1} - L_{j,t-1})}} - \frac{\sum_{i=1}^{N} e^{\eta(-L_{i,t-1})}}{\sum_{i=1}^{N} e^{\eta(-L_{j,t})}}$$

The main advantage of this approach [11] [14] is that it does not depend on the past predictions of the forecaster ($p_s, s < t$), but only on the past performance of the experts. Moreover, the weights are easy to compute as

$$w_{i,t} = \frac{w_{i,t} e^{-\eta l(\acute{f}_{i,t})}}{\sum_{j=1}^{N} w_{j,t-1} e^{-\eta l(\acute{f}_{j,t})}}$$

2.3 Limited Feedback Models

The Exponentially Weighted Average Forecaster points to the necessity for the development of the Bandit class of models. We can define classes of problems where the forecaster has access to limited past information, most notably not knowing the true outcome $\hat{}_t$ after each iteration. Here we will focus on a few important variants which will introduce the strengths and weaknesses of Bandit models, which are designed to operate under limited information and still mitigate losses.

2.3.1 Label Efficiency Prediction

This is the simplest case of limited feedback, where the forecaster can make predictions but have access to only a subset of the outcomes. Formally [11]:

Parameters:

- N: number of actions
- I: action space
- Y: outcome space
- l: loss function
- $\mu : N \to N$: query rate

Then, for each round $t = 1,2,\ldots$

1. The environment selects the next outcome $\hat{y}_t \in Y$, but does not disclose it to the learner.
2. The forecaster selects an action $I_t \in \{1,2,\ldots,N\}$
3. The forecaster receives a loss $l(I_t, \hat{y}_t)$, and each action i also incurs a loss $l(i, \hat{y}_t)$, but none of these loss values are disclosed to the forecaster.
4. If fewer than $\mu(t)$ queries have been made up to time t, the forecaster is allowed to submit a new query to observe the outcome $\hat{y}_t$. Otherwise, if no query is made, the outcome remains hidden.

The goal of the forecaster is to minimize the difference

$$L_n - min_{i=1,2,,,N} L_{i,n} = \sum_{t-1}^{n} l(\acute{I}_{t,t}) - min_{1,2,,,N} \sum_{t-1}^{n} l(\acute{\iota}_t)$$

Obviously, central to the forecaster's strategy is the way they choose whether to make a query for$\hat{}_t$ or not. To achieve non-trivial performance, a simple biased coin approach can be sufficient.

Parameters:

- $\eta > 0$
- $0 \leq \varepsilon \leq 1$

Initialisation:

- $w_0 = (1,1,1,\ldots,1)$

Then, for each round $t = 1,2,\ldots$

1. Draw an action I_t from $\{1,2,\ldots,N\}$ according to the distribution

$$p_{i,t} = \frac{w_{i,t-1}}{\sum_{j=1}^{N} w_{j,t-1}}$$

i=1,2,,,

2. Draw a Bernoulli random variable Z_t such that $P[Z_t = 1] = \varepsilon$

3. If $Z_t = 1$, then obtain$\hat{}_t$ and compute

   $w_{i,t} = w_{i,t-1}e^{-\frac{\eta l(\hat{I}_t)}{s}}$ for each $i = 1,2,\ldots,N$

   else $w_t = w_{t-1}$

2.3.2 Partial Monitoring in General

The limitation which partial monitoring answers is the inability of the forecaster to know the losses they have suffered in the past. In this case, the forecaster is privy to a function $h(I_t\hat{}_t)$ which conveys information on the outcome of each pair $I_t\hat{}_t$, without allowing access to each of these values individually. Formally [11]:
Parameters:

- $Y = \{1,2,\ldots,M\}$: outcome space
- N: number of actions
- l: loss function
- h: feedback function

Then, for each round $t = 1,2,\ldots$

1. The environment selects the next outcome $\hat{y}_t \in Y$ but does not disclose it directly.
2. The forecaster experiences a loss $l(I_t, \hat{y}_t)$, and every possible action i also incurs a loss $l(i, \hat{y}_t)$; however, none of these losses are revealed to the forecaster.
3. The forecaster suffers a loss $l(I_t, \hat{y}_t)$, while every action i incurs its own loss $l(i, \hat{y}_t)$. However, these losses remain hidden from the forecaster. Instead, the only available information is the feedback signal $h(I_t, \hat{y}_t)$.

2.3.3 Multi Armed Bandit

This class of problems is defined by the forecaster being unable to know the outcomes of any action other than the one they took. This scenario represents a specific instance of the partial monitoring framework, where it is typically assumed that the feedback provided to the forecaster corresponds directly to their incurred loss. The goal is still to make sure that the cumulative loss of the forecaster $\sum^{n} l(I_t, \hat{t})$ is not much larger than the cumulative loss of the best expert, namely $min_{i=1,2,..,N} \sum_{t=1} (i, \hat{t})$ [11].

The classical formulation of the problem assumes that the losses are randomly and independently drawn from an unknown, fixed distribution. This formulation allows for a two-step strategy [50]. In the exploration phase, all "arms" are sampled in order to estimate the means of the loss distributions. This is followed by the exploitation phase, where the forecaster chooses the action with the smallest estimated loss. This approach requires a high level of confidence in the sharpness of the means identified during exploration, and the optimization issue refers to the tradeoff between exploration and exploitation (as the exploration itself brings losses).

A special case of this problem is when the outcomes $\hat{}_t$ are generated in a non-oblivious way, a variant called the non-stochastic, or adversarial multi-armed bandit model [11]. In this variant, the forecaster's actions cannot depend on past values of $l(i, \hat{t})$, except when $i = I_t$. At this point, we introduce the following notation to denote gains instead of losses:

$$g(i, \hat{t}) = 1 - l(i, \hat{t})$$

$g(i, \hat{}) = \frac{g(i_t)}{p_t}$ if $I_t = i$, or $g(i, \hat{}) = 0$ otherwise

An improved strategy for the bandit algorithm is as follows:

Parameters:

- N: number of actions
- $\beta, \eta, \gamma \leq 1$: positive real numbers

Initialisation:

- $w_{i,0} = 1$
- $_{i,1} = \frac{1}{N}$ for $i = 1,2,\ldots,N$

Then, for each round $t = 1,2,\ldots$

1. Draw an action I_t from $\{1,2,\ldots,N\}$ according to the probability distribution p_t
2. Calculate the estimated gains

$$g'(i,\hat{}) = g(i,\hat{}) + \frac{\beta}{} = \frac{g(\hat{i},t)+\beta}{} \text{ if } I = i \text{ or } t$$

$$g'(i,\hat{}) = g(i,\hat{}) + \frac{\beta}{} = \frac{\beta}{} \text{ otherwise}$$

3. Update the weights

$$w_{i,t} = w_{i,t-1}e^{\eta g'(\hat{i},t)}$$

4. Calculate the updated probability distribution

$$p_{i,t+1} = (1-\gamma)\frac{w_{i,t}}{W} + \frac{\gamma}{N},\ i = 1,2,\ldots,N$$

In the work of Slivkins [47], attention is drawn to the fact that the aim in an adversarial bandit problem with expert advice is not to identify the best action in general, but rather to minimize regret relative to the best expert. The following protocol is proposed:

Parameters:

- K: arms
- E: set of N experts
- T: number or rounds

For each round $t = 1,2,\ldots$

1. Adversary picks the cost $c_t(\alpha) \geq 0$ for each arm $\alpha \in K$
2. Each expert $e \in E$ recommends an arm $\alpha_{t,e}$ (observed by the algorithm)
3. The algorithm picks the arm $\alpha_t \in K$ and receives the corresponding cost $c_t(a_t)$

The total cost of each expert is defined as

$$cost(e) = \sum_{t=1}^{T} c_t(\alpha_{t,e})$$

and the goal is to minimize $R(T) = cost(forecaster) - min_{\in E} cost(e)$

We note that the solution given in this case corresponds to experts whose recommendations $\alpha_{t,e}$ are chosen in advance, thus they cannot learn over time.

2.3.4 The Exp3 Algorithm

A simple algorithm proposed in the work of T. Lattimore and C. Szepesvári [37] for the adversarial bandits problem leverages exponential weighting in the probability distribution used by the forecaster. This solution assumes an oblivious adversary, in other words the adversary chooses rewards at the start of the game. In contrast, a non-oblivious (or reactive) adversary has access to previous rewards and actions ($x_1, A_1, x_2, A_2, \ldots, x_{t-1}, A_{t-1}$ respectively) and can have x_t depend on them. For the former, simpler case, the Exp3 algorithm is as follows:

Parameters:

- n: horizon
- k: number of actions
- η: learning rate

Initialisation:

- $\hat{}_i = 0$ for all i

Then, for each round $t = 1,2,\ldots$

1. Calculate the sampling distribution

$$p_t = \frac{e^{\hat{\eta}_{t-1,i}}}{\sum_{j-1}^{k} e^{\hat{\eta}_{t-1,j}}}$$

2. Sample $A_t \sim p_t$ and observe the reward X_t
3. Calculate the total estimated reward $\hat{}_{t,i}$

$$_{t,i} = \hat{}_{t-1,i} + 1 - \frac{I\{A_t = i\}(1 - X_t)}{p_{t,i}}$$

2.3.5 The Exp3-IX Algorithm

This is a modification of the Exp3 algorithm [37], in order to ensure a small expected regret, which is well-concentrated around its mean value. Formally:

Parameters:

- n: horizon
- k: number of actions
- η: learning rate
- γ: independently defined constant

Initialisation:

- $L_i = 0$ for all i

Then, for each round $t = 1,2,\ldots$

4. Calculate the sampling distribution

$$p_{t,i} = \frac{e^{-\eta L_{t-1,i}}}{\sum_{j-1}^{k} e^{-\eta L_{t-1,j}}}$$

5. Sample $A_t \sim p_t$ and observe the reward X_t
6. Calculate the total estimated reward $\hat{}_{t,i}$

$$L_{t,i} = L_{t-1,i} + 1 - \frac{I\{A_t = i\}(1 - X_t)}{p_{t,i} + \gamma}$$

This change produces a p_t which assigns non-negligible probability to actions with large losses. This ensures that arms which would be virtually always overlooked by the Exp3 are still explored occasionally with Exp3-IX.

3. “Adaptive Market Making via Online Learning”: Solid Groundwork for Implementation

3.1 Introduction

This section outlines the experimental framework and methodology employed to assess model performance. Building upon the foundational work of Abernethy and Kale in "Adaptive Market Making via Online Learning" (2013) [1], our approach adopts their formulation of a dynamic, strategy-driven market-making system, in which multiple trading strategies—referred to as “experts”—are evaluated and combined using regret-minimizing online learning algorithms. Following a concise theoretical background, we describe the practical tools and implementation choices that guide the empirical validation of this model.

For experimental purposes, we simulate a market involving a single tradable asset, with its price at time t denoted by p_t. To simplify computations, we set the discretization parameter δ to 1. Historical stock data is retrieved through the Yahoo! Finance API, adhering to constraints detailed later in the section.

At the beginning of each time step t, a trading strategy maintains two key state variables: its inventory H_t, representing the number of shares held (positive for long positions, negative for short), and its cash position C_t, which tracks cumulative profit and loss. The strategy is initialized with no holdings or cash ($C_1 = H_1 = 0$), and its total portfolio value is defined by $V_{t+1} = C_{t+1} + p_t H_{t+1}$.

The function L(p) describes the number of shares the strategy intends to buy or sell at price p. We consider two types of orders in our simulation:

A market order is executed immediately at the current price p_t for X shares. After execution, the cash position is updated as $C_{t+1} = C_t - p_t X$, and holdings become $H_{t+1} = H_t + X$. When X is negative, the trader has sold shares short, which results in an increase in cash.

A limit order allows trades to be placed at specified prices and remains in effect until matched. We assume that shares are transacted for any price between p_{t-1} and p_t. If prices increase ($p_t > p_{t-1}$), then for each price level p in the range (p_{t-1}, p_t], the cash is updated as $C_{t+1} = C_t + pL_t(p)$, and the inventory decreases by the corresponding quantity. If prices decrease ($p_t < p_{t-1}$), then for $p \in [p_t, p_{t-1})$, the cash decreases and holdings increase accordingly.

3.2 Strategies

3.2.1 Spread Based Strategy

This strategy selects a price window $[a_t, a_t + b]$ during each round t. For a fixed liquidity density parameter a, it submits a buy and a sell order within this window. Depending on the price p_t, the window is adjusted for the next round by the smallest amount to include p_t. Formally:

Input:

- $b > 0$
- $a > 0$: liquidity density
- p_1: initial price

Initialization:

- $a_1 = p_1$

For $t = 1,2,\ldots,T$:

1. Observe the market price p_t
2. If $p_t < a_t$ then $a_{t+1} \leftarrow p_t$
3. Else if $p_t > a_t + b$ then $a_{t+1} \leftarrow p_t - b$
4. Else $a_{t+1} \leftarrow a_t$
5. Submit limit order L_{+1}:
   If $p \in [a_{t+1}, a_{t+1} + b]$ then $L_{t+1}(p) = 0$
   Else $L_{t+1}(p) = a$

The trading here is, therefore, done by maintaining a moving window of acceptable prices at every step. It uses this window to decide where to place buy and sell orders. At each round, the strategy defines a price range based on where the market currently is. If the latest price falls outside of the window, the window shifts just enough to include that price. If the price is already within the window, the window stays the same.

Then, using a fixed parameter that defines how much volume to trade, the strategy places limit orders: it avoids placing orders within the current window to reduce the chance of immediate execution, and instead places them outside the window where the price is more favorable.

The approach balances passive liquidity provision with responsiveness to market changes. When prices shift too much, the strategy adapts by shifting the interval accordingly, ensuring that the limit orders remain relevant.

3.2.2 Low Regret Meta-Algorithm

Formally:

1. Execute each strategy$\hat{}(b)$ in parallel so that at the end of each t, all trades are carried out, allowing for the calculation of the updated vectors H_{t+1}, C_{t+1} and $V_{t+1} \in R$ can be executed.

2. Initialize a regret-minimizing algorithm A designed for the expert advice setting, assigning one expert to each strategy$\hat{}(b)$ for $b \in B$. Let the distribution over strategies generated by A at time t be w_t.

3. For $t = 1,2,\ldots,T$ do

    a. Carry out any outstanding market orders from the previous time step using the current market price p_t so that the inventory now equals $H_t\ w_t$. The cash value changes by $-(H_t(\ w_t - w_{t-1}))\ p$.

    b. Carry out any outstanding limit orders from the previous time step: a w_t weighted blend of the limit orders of the strategies $S(b)$. The holdings change to $H_{t+1}\ w_t$, and the cash value changes by $(\ C_{t+1} - C_t)\ w_t$.

c. For each strategy$\hat{}(b)$ for $b \in B$, set its payoff in round t to be $V_{t+1}(b) - V_t(b)$ and send these payoffs to A.

d. Obtain the updated distribution w_{t+1} from A.

e. Submit a market order to buy $H_{t+1}(w_{t+1} - w_t)$ shares in the upcoming time step, and a w_{t+1} weighted combination of the limit orders of the strategies $S(b)$.

The model operates by running all strategies in parallel so that, after each time step, all their trades are completed and the updated values for holdings, cash, and overall portfolio value can be calculated. A regret-minimizing learning algorithm is initialized, where each strategy is treated as a distinct expert. This algorithm generates a probability distribution over the strategies, reflecting their relative performance.

At each time step, any outstanding market orders from the previous round are executed using the current market price. This results in an adjustment of the portfolio's inventory and cash balance. Similarly, any remaining limit orders from the previous round are executed based on a weighted combination of each strategy's suggested actions, which again updates both the inventory and cash values.

The performance of each strategy is then evaluated by measuring how much its total portfolio value changed during the current round. These payoffs are sent back to the learning algorithm, which uses them to update the strategy distribution going forward.

Finally, the model places new trades for the next round: market orders based on the change in the distribution over strategies, and limit orders formed from the newly updated strategy mixture. This cycle continues at each time step, allowing the algorithm to adaptively follow the most successful strategies over time.

3.2.3 Multiplicative Weights

Input:

- η_t for t = 1,2, . . . , T:

Initialisation:

- $w_1(b) = \frac{1}{N}$ for $b \in B$

For $t = 1,2, \ldots, T$:

1. Choose an action based on the probability distribution w_t (normalized so that $\sum w_t = 1$)
2. The costs of the decision are revealed
3. Calculate $w_{t+1} = w_t \frac{e^{\eta_t(V_{t+1}(b) - V_t(b))}}{t}$, where Z is the normalization constant to make w_{t+1} a distribution.

The algorithm begins by initializing the weights assigned to each strategy uniformly. At each time step, it selects an action by sampling from the current probability distribution over strategies. This distribution is normalized so that all weights sum to one. Once an action is taken, the resulting costs or outcomes are observed. Using this feedback, the algorithm updates the weight distribution for the next time step. Specifically, each strategy's new weight is adjusted based on how much its portfolio value changed, scaled by a learning rate parameter. These updates are performed exponentially, giving higher weight to strategies that performed better in the most recent round. A normalization constant ensures the resulting weights again form a valid probability distribution. This dynamic adjustment allows the algorithm to increasingly favor better-performing strategies over time.

3.2.4 Follow the Perturbed Leader

Next, we present MMFPL (Market Making using Follow-The-Perturbed-Leader).

Input:

- N: the number of strategies
- T: the time limit of the algorithm

Initialization:

- $\eta = \frac{1}{2G}\sqrt{\frac{log(N)}{T}}\, b \in B$

For every $b \in B$, let $p(b)$ be a sample from the exponential distribution with mean 1/n.
The distribution wt is then updated to match that of the "perturbed leader," reflecting the strategy that performs best after applying random perturbations to past losses.

$$w_t(b) = Pr[\, V_t(b) + p(b) \;\geq\; V_t(b') + p(b') \;\forall\; b' \;\in B]$$

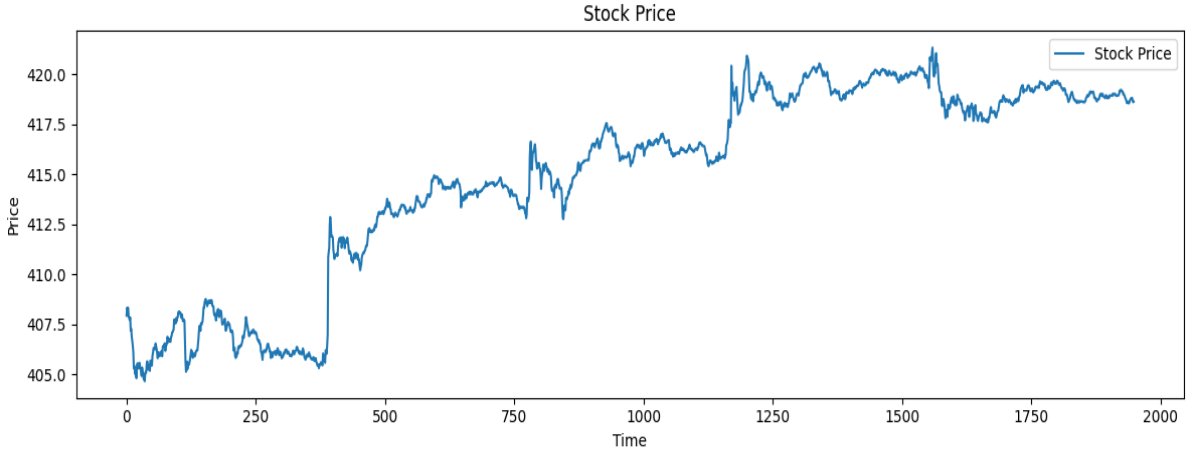
The MMFPL algorithm, which stands for Market Making using Follow-The-Perturbed-Leader, takes as input the total number of strategies and a predefined time horizon for which the algorithm will operate. Initially, a learning rate is set, typically chosen as the inverse square root of the logarithm of the number of strategies, ensuring a controlled sensitivity to performance differences.

For each available strategy, a random value is drawn from an exponential distribution with a mean of one. These sampled perturbations are added to the historical performance of each strategy. The algorithm then selects a strategy distribution by identifying which perturbed leader—the strategy with the best performance after adding the random noise—appears optimal. Formally, the probability assigned to each strategy corresponds to the likelihood that its perturbed performance exceeds that of all others. This randomized selection introduces exploration into the decision-making process, helping the algorithm remain adaptive in uncertain or shifting environments.

3.3 Implementation and Observations

In this next section we are going to implement the model and discuss result s based on historical data for the Microsoft stock (ticker: "MSFT"), around a random period of 8 consecutive days, with 1-minute intervals.

Observations on Stock Price Behavior

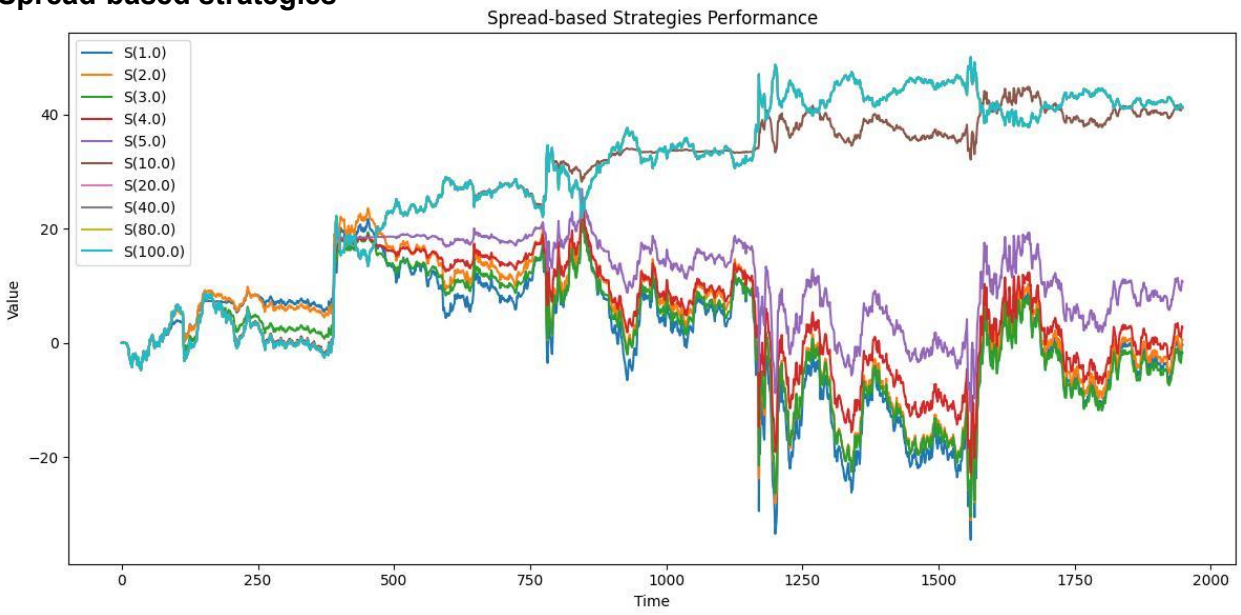


[36]

The plotted stock price data shows a generally upward trend over time, starting around $405 and reaching above $420 by the end of the period. While the overall direction is positive, the price series includes noticeable jumps at several points, such as around time steps 400, 750, and 1,150, suggesting sudden market responses—possibly to news or other external events.

Fluctuations in price indicate typical intraday volatility, and the presence of both gradual trends and abrupt changes highlights the non-stationary nature of the data. These characteristics are a clear support for the use of time-sensitive adaptive strategies like online learning algorithms, which are better suited for changing market conditions compared to static approaches.

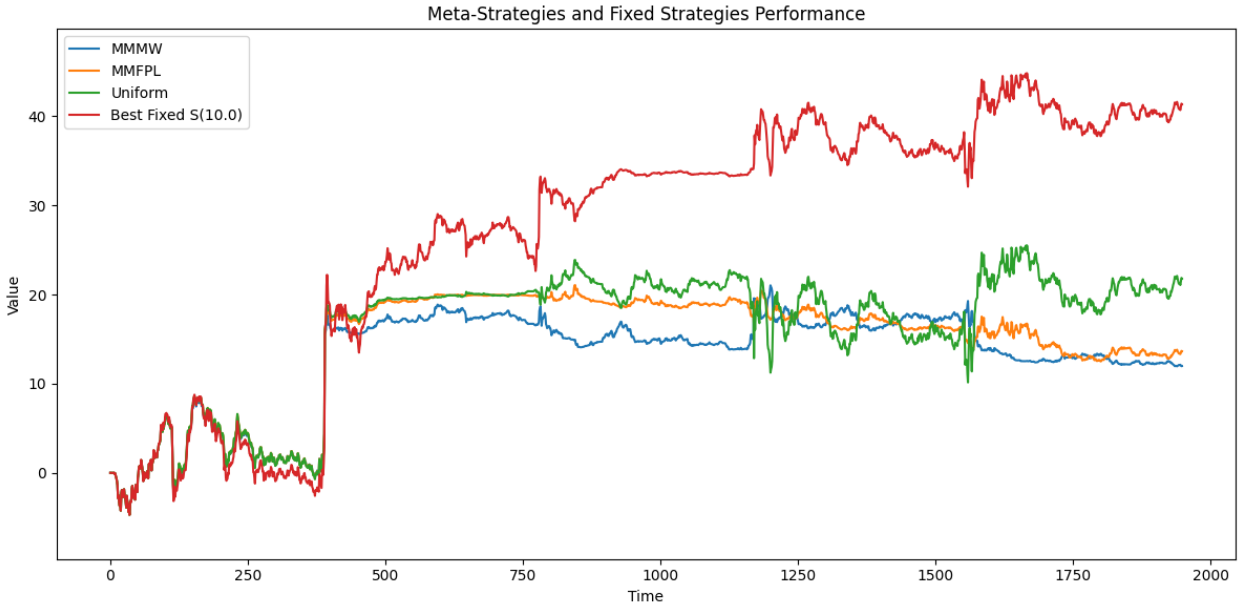
Spread-based strategies



The plot compares the performance of spread-based trading strategies using different window sizes (b). Strategies with larger windows (e.g., b = 10,100) tend to perform better overall, showing higher and more stable cumulative values throughout the trading period. In contrast, strategies with smaller b values (e.g., b = 1 to 5) exhibit greater volatility and lower overall returns, often dropping into negative territory.

This suggests that broader historical context (i.e., longer window sizes) helps these strategies make more stable and profitable decisions, likely by reducing sensitivity to short-term noise in the price series.

Final Strategy Performance Analysis



The plot compares three meta-strategies—MMMW (Multiplicative Weights), MMFPL (Follow-the-Perturbed-Leader), and Uniform—against the Best Fixed Spread-Based Strategy (with window size b=10).

Observations

1. **Best Fixed Strategy, S(100.0)**:

- The best-performing spread-based strategy significantly outperforms all meta-strategies.
- It shows a strong upward trend throughout the period, achieving a final portfolio value well above 40.
- This suggests that if the optimal fixed strategy could be selected in hindsight, it would deliver the highest return.

2. **Uniform Strategy**:

- Performs second best overall, displaying a relatively stable upward trend after an initial dip.
- Its stability stems from averaging across all spread-based strategies, which buffers against the volatility of any single one.
- The uniform strategy captures some gains from high-performing strategies like S(10.0), without suffering too much from poor performers.

3. **MMMW (Multiplicative Weights)**:

- Starts out with modest gains but eventually stagnates and slightly declines in later periods.
- Despite being a theoretically sound regret minimization algorithm, MMMW appears conservative in this setting, possibly due to over-hedging or under-allocating to higher-risk strategies.

4. **MMFPL (Follow the Perturbed Leader)**:

- Exhibits more variability than MMMW and ends with the lowest cumulative value among all strategies.
- This result may be attributed to the stochastic nature of perturbations, which can occasionally steer the algorithm toward less effective strategies.

In this chapter, we evaluated the performance of several spread-based strategies and meta-strategies. While fixed strategies such as S(100.0) and S(10,0) outperformed others in terms of empirical returns, meta-strategies like the Multiplicative Weights (MMMW) fell short in practice, underperforming both the best fixed strategy and even the uniform ensemble.

However, MMMW remains of central focus for our continued analysis. Despite the suboptimal empirical performance, this algorithm is one of the most well-established frameworks in online learning. This method has strong theoretical guarantees and serves as a foundation for a wide range of online learning algorithms in adversarial and dynamic environments.

Because of its theoretical richness and prominence in the online learning literature, MMMW offers significant potential for deeper exploration and meaningful refinement. This also aligns closely with the core objective of this thesis: to understand and enhance the application of online learning algorithms in (adaptive) market-making.

Accordingly, the next chapter will focus on meta-strategies, namely the MMMW. We aim to dissect it's structure, identify the reasons behind the ill performance, and explore ways to try and make it more robust and responsive in real-world trading scenarios.

4.Alternate Approach and Theoretical Comparison

In this final chapter, we propose an enhancement to the MMMW algorithm by relaxing a fundamental assumption inherent in classical online learning frameworks. Traditional online learning algorithms typically treat all past observations with equal importance, implicitly assuming a stationary environment in which the underlying loss distribution remains stable over time and the best-performing expert is consistently relevant throughout the entire horizon.

However, this stationarity assumption can be problematic in real-world, time-sensitive applications, where sudden structural shifts—regime changes—can lead to significant performance degradation. In such settings, failing to adapt to the evolving nature of the environment may result in substantial capital losses and sharp increases in regret.

Several concrete examples illustrate the nature of these regime changes:

- A financial asset may exhibit trending behavior before transitioning abruptly into a mean-reverting regime;
- Markets may shift from a low-volatility state to one characterized by high volatility;
- Liquidity conditions may deteriorate unexpectedly, changing from a liquid to an illiquid state.

These regime shifts can occur without warning, rendering static learning strategies inadequate. In response, this chapter introduces a modification to the MMMW algorithm that incorporates mechanisms for detecting and adapting to non-stationarity. The proposed enhancement aims to improve robustness and responsiveness in dynamic environments, thereby mitigating the adverse effects of regime shifts on algorithmic performance.

4.1 Laying the Theoretical Groundwork

To address the above we introduce the idea of "memory" into the algorithm. More specifically, we apply a discounting or decay mechanism that gives more weight to recent performance and gradually reduces the influence of older data. This is achieved by introducing a discount factor, which we denote by $\gamma \in (0,1]$. At each step, past losses (or rewards) are multiplied by γ, making their impact on the current decision smaller as time progresses.

This leads to a variant of the algorithm we may call **Discounted Multiplicative Weights (DMW)**. In DMW, instead of keeping a total of all past losses, we maintain a discounted running total that reflects how recent each observation is. The weight updates still follow the familiar exponential rule of MW but are now based on this time-weighted loss. As a result, the algorithm can respond more quickly to changing environments, adapting to new trends and behaviors while naturally forgetting outdated patterns.

In the rest of this work, we define this approach formally, explain how it differs from standard MW, and demonstrate through examples and experiments why incorporating memory through discounting is beneficial in non-stationary settings. We will be heavily relying on Abernethy and Kale's work "Adaptive Market Making via online learning" as well as citations, lemmas and theorems used broadly in the paper.

Simple Interpretation

- As $\gamma \to 1$, converge to standard MW
- As $\gamma \to 0$, consider the most recent loss

The decay factor γ controls memory, "rule of thumb":

- $\gamma \approx 1.0 \to$ long memory, like classic MW
- $\gamma \approx 0.7–0.9 \to$ moderate memory
- $\gamma \approx 0.3–0.5 \to$ very short memory, high responsiveness

We will now introduce some basic changes in the original mathematical formulas after remembering the core ones from the initial model.

Input:

- Time Horizon: T
- N experts, $i \in \{1, \dots, N\}$
- At each round t = 1,2,...,T:
  - Learner chooses a distribution $w \in \boldsymbol{\Delta^N}$
  - Environment reveals a loss vector $l_t \in [\mathbf{0}, \mathbf{1}]^{\boldsymbol{N}}$
  - Learner incurs loss: $w^T l_{tt} = \sum_{i=1}^{N} w^{(i)} l_t^{(i)}{}_t$

Classic MMMW Update Rule

Cumulative loss up to time t:

$L^{(i)} = \sum_{T=1}^{t} l^{(i)}{}_T$

Exponential weight update:

$$W_{t+1}^{(i)} = \frac{e^{(-\eta L_t^{i})}}{\sum_{j=1}^{N} (-\eta L^{(j)}) t}$$

DMW Update Rule

We now introduce the discount factor $\gamma \in (0,1]$. Changes are made accordingly to the formulas:

DMW Cumulative loss:

$L^{(i)} = \gamma L_{t-1}^{i} + l^{(i)}{}_t L^{i)} = 0, \; L^{i)} = \sum_{t=1} \gamma^{T-t} l^{(i)}{}_T$

Each new loss is added, but older losses shrink by γ every round.

DMW Weight update:

$$w_{t+1}^{(i)} = \frac{e^{(-\eta L_t^{i})}}{\sum_{j=1}^{N} e^{(-\eta L^{(j)})}} .$$

4.2 Description of the enhanced algorithm

The DMW algorithm builds on the classic MMMW but adjusts how it keeps track of each expert's performance over time.

Here’s how it works in practice:

1. Initialize weights:
   Start with equal weights for all experts. This reflects total uncertainty at the beginning.

2. Observe expert performance:
   In each round, after choosing a probability distribution over experts, receive feedback typically, a loss vector (how poorly each expert performed).

5. Apply discount to history:

   Instead of storing the full cumulative loss for each expert, maintain a discounted total, where older losses shrink over time. This is done using a discount factor γ.

6. Update weights using exponential penalty:

   Use the discounted cumulative loss in the standard MW exponential update as described previously.

   Experts that recently performed poorly lose weight more quickly, while recent strong performers gain weight faster.

5. Normalize weights:

   Ensure the weights still form a valid probability distribution (they sum to 1).

6. Repeat:

   At the next step, repeat the process using the updated weights and newly observed losses.

Pseudocode

Initialize:

N = number of experts
η = learning rate
γ = discount factor
weights = [1/N] * N (vector, length N)
discounted losses = [0.0] * N

For t = 1 to T:

1. Select a probability distribution over experts:
   w = normalize(exp(-η * discounted losses))

2. Observe loss vector: loss = $[l_1, l_2, \ldots, l_N]$

3. Update discounted losses:
   for i in 1 to N:
      discounted losses[i] = γ * discounted losses[i] + loss[i]

4. Recompute weights:
   for i in 1 to N:
      unnormalized weight[i] = exp(-η * discounted losses[i])
   weights = normalize(unnormalized weight)

4.3 Addressing Regret

Recap

As mentioned regret is the difference between the learner's cumulative loss and that of the best expert:

$$Regret_T = \sum_{t=1}^{T} w^T l_t - min \sum_{t=1}^{T} l^{(i)}{}_t$$

As with all the components of the classic MMMW approach this assumes all time steps are equally important. This doesn't fit the non stationary, suddenly fluctuant enviroment we are challenging in this chapter.

Introduction of Discounted Regret

To model the idea that recent time periods matter more, we define discounted regret using the known discount factor $\gamma \in (0,1]$:

- Discounted Regret of expert i:

$$Regret_T^{\gamma} = \sum_{t=1}^{T} \gamma^{T-t} \left(w_t^T l_t - l_t^{(i)}\right)$$

- And Overall Discounted Regret:

$$Regret_T^{\gamma} = max_i Regret_T^{\gamma}(i)$$

Respectively as before:

- The term γ^{T-t} provides more weight to recent rounds, fading out older periods

- This regret compares the learner to each fixed expert, but only over recent effective memory

Intuitively:

- As $\gamma \to 1$, we recover to standard MMMW regret.

- As $\gamma \to 0$, only the final time step matters.

This newly modeled regret will enable our experimental algorithm to be evaluated fairly in shifting settings.

Summary and Intuitive Commentary

In many real-world applications — especially in financial markets — the environment is rarely stationary. Market conditions change: a strategy that performs well during a volatile phase may perform poorly when the market becomes calm, or vice versa. These regime changes make it dangerous for an algorithm to treat all historical information as equally important.

The standard Multiplicative Weights (MW) algorithm maintains cumulative losses for each expert**, reweighting** them based on all past performance. This is effective in stable settings, but it causes the algorithm to adapt slowly when recent data contradict long-term trends.

To address this, we introduce a discount factor $\gamma \in (0,1]$ into the MW algorithm. Rather than summing past losses equally, we compute a discounted cumulative loss, which emphasizes

more recent rounds and diminishes the influence of older ones:

$$\acute{\boldsymbol{L}}_{\boldsymbol{t}}^{(\boldsymbol{i})} = \boldsymbol{\gamma}\acute{\boldsymbol{L}}_{\boldsymbol{t-1}}^{\boldsymbol{i}} + \boldsymbol{l}_{\boldsymbol{t}}^{(\boldsymbol{i})}$$

This results in a Discounted Multiplicative Weights (DMW) algorithm, where expert weights are updated based on:

$$\boldsymbol{w}_{\boldsymbol{t+1}}^{(\boldsymbol{i})} = \boldsymbol{e}^{-(\boldsymbol{\eta}\acute{\boldsymbol{L}}_{\boldsymbol{t}}^{(\boldsymbol{i})})}$$

The learner now relies more on recent performance when making decisions, which helps in dynamic environments.

Furthermore, instead of evaluating performance over the full history, we define discounted regret as:

$$Regret_T^{\gamma} = \sum_{t=1}^{T} \gamma^{T-t}\left(w_t^T l_t - l_t^{(i)}\right)$$

This expression compares the algorithm to each fixed expert, but places more weight on recent time steps. The result is a regret measure that reflects how well the learner is keeping up with the current best strategy, rather than the historically best one.

While we do not provide a formal proof here, this approach is motivated by known theoretical results for online learning, used in Abernethy and Kale's paper "Adaptive Market Making via online learning", where regret bounds grow proportionally to an effective window size $\sim \frac{1}{1-\gamma}$ [11]. This provides theoretical justification for introducing memory decay: it allows the learner to adapt more quickly to recent changes, at the cost of **higher sensitivity to noise**.

It is important to note that while incorporating memory through discounting allows the algorithm to respond more rapidly to sudden shifts or regime changes in the market, it also increases the model's sensitivity to stochastic noise. This trade-off between adaptability and stability is a fundamental characteristic of online learning methods. In the next and final section of this study, we address this issue and propose methods to mitigate the impact of noise while preserving the benefits of responsiveness.

4.5 Conclusion

The primary objective of this study was to explore recent developments in the domain of market making through the lens of online learning algorithms. To this end, we began by introducing the concept of market making and outlined how the problem is perceived from two distinct but complementary perspectives: quantitative finance and computer science. We then provided an overview of algorithmic approaches commonly employed in the field, with a particular emphasis on expert-based models. Finally, we proposed an enhancement to a well-established method—namely, the Multiplicative Weights (MMMW) algorithm—by incorporating memory through discounting, aiming to improve adaptability in dynamic market environments.

The proposed enhancement to the MMMW algorithm, through the introduction of memory via discounting, is particularly useful in the context of financial markets due to the non-stationary nature of real-world environments. Market conditions are subject to frequent and sometimes abrupt changes—shifting from trending to mean-reverting behaviors, or from calm to volatile regimes. Traditional online learning methods that weigh all past observations equally may react too slowly to these changes, resulting in suboptimal decision-making.

By incorporating a discount factor, the algorithm emphasizes recent performance while gradually forgetting outdated information. This enables it to adapt more quickly to regime shifts, improving both responsiveness and robustness in dynamic settings. The method remains general and computationally efficient, preserving the interpretability and simplicity of the original Multiplicative Weights framework while offering significant improvements in adaptability.

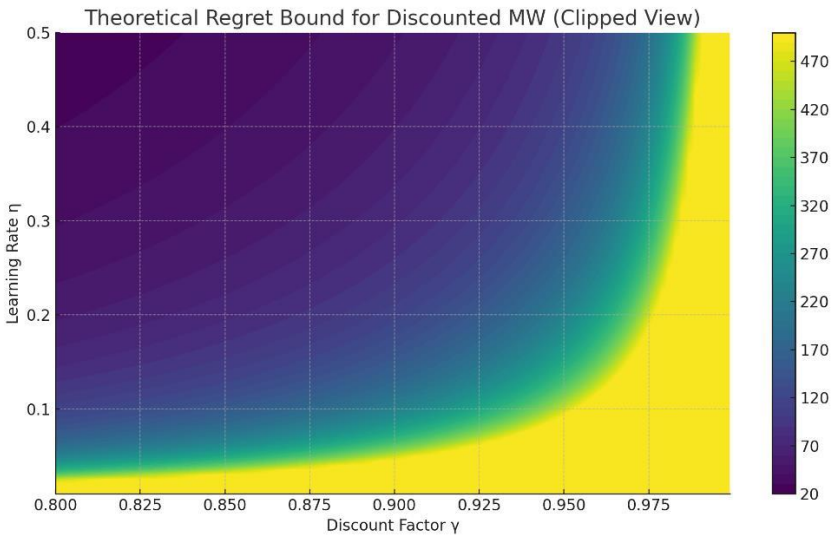
As a result, this approach provides a practical and theoretically motivated solution for real-time strategy allocation in algorithmic trading and beyond, especially in environments where recent observations are more predictive than long-term averages.

While the introduction of discounting improves the algorithm's ability to adapt to changing environments, it also introduces notable limitations—chief among them, increased sensitivity to noise.

By prioritizing recent observations, the algorithm becomes more reactive not only to genuine regime shifts but also to short-term fluctuations that may be purely stochastic. This **hypersensitivity** can lead to frequent over-adjustment of strategy weights in response to transient, non-informative market movements. In practical terms, this may manifest as unstable behavior, excessive trading, or overfitting to noise, particularly in highly volatile environments. Furthermore, the performance of the algorithm becomes increasingly dependent on the choice of the discount factor γ. If γ is set too low, the algorithm may disregard valuable historical context; if

it is too high, the benefits of adaptability are diminished. Selecting an appropriate γ, along with the learning rate η is extremely crucial.

4.5.1 Discount and Learning Rate trade off



Finally, while the discounted model is computationally efficient and retains the simplicity of the original framework, it still lacks the capacity to explicitly detect or model structural changes in the data. Thus, in practice, its success relies on carefully balancing responsiveness and stability and possibly supplemented by additional noise-smoothing or change-detection mechanisms.

The interaction between the discount factor γ and the learning rate η plays a central role in determining the performance of the Discounted Multiplicative Weights algorithm. When the discount factor is low—that is, when the algorithm relies on a short memory—it naturally emphasizes recent observations and becomes more responsive to changing conditions. In such settings, a moderate learning rate η is typically sufficient to ensure low regret, as the algorithm adjusts quickly while still maintaining a level of stability.

However, as the discount factor γ increases—corresponding to longer memory—the algorithm begins to behave more like its classical, non-discounted realative. In this regime, if the learning rate is not adjusted accordingly, the algorithm can suffer from **spiking regret**. This occurs because long memory causes outdated and potentially misleading losses to accumulate, and a large η may then overreact to these aggregate values, especially in volatile environments. The result is a trade-off in the *γ,η* parameters, with what is known as a narrow ridge where performance is optimal. Along this ridge, the algorithm balances memory and reactivity: as

memory increases (larger γ, the learning rate must decrease (smaller η)) to maintain stability and prevent overreaction. Conversely, shorter memory allows for more aggressive learning without compromising robustness. Understanding and exploiting this trade-off is essential for tuning the algorithm effectively in practice. Thus:

- If $\gamma \uparrow$, then $\eta \downarrow$
- If $\gamma \downarrow$, then $\eta \uparrow$.

Ideally, we'd like the regret surface to show something similar to the below rigde, a stable subspace in in the (γ,η) space:

4.5.2 Limit Order Book Simulation

The most common and effective practice in the study of online learning algorithms—particularly within market-making and trading—is to begin with experiments on synthetic data before transitioning to real-world financial data. Synthetic data allows researchers to fully control the environment, explicitly define regime changes, and establish a known best-performing expert or strategy. This level of control is critical for isolating the effects of algorithmic modifications, such as discounting or adaptivity, and for understanding how the algorithm behaves under specific market conditions. Once the model's responsiveness and stability are well-understood in a simulated setting, it can then be tested on actual market data, where challenges such as noise, structural breaks, and unpredictable behavior are far more complex. This staged approach—starting from the controlled and moving toward the realistic—is standard in the field and ensures that observed effects are due to model behavior rather than incidental properties of the dataset [47] [4].

In this study, we should first rely on synthetically generated data to evaluate the behavior of online learning algorithms in a controlled market-making environment. Importantly, this synthetic dataset does not require any real-world (primary) financial data as input. Instead, we construct the price series using well-established stochastic models from financial mathematics, such as geometric random walks, mean-reverting processes, and high-volatility noise sequences [4]. These models allow us to emulate distinct market regimes — including trending periods, mean-reverting dynamics, and volatile bursts — by simply varying parameters like drift, volatility, and reversion strength.

For example, trending markets can be simulated using a price update of the form $p_{+1} = p + \mu + \varepsilon_t$, where μ is a fixed drift (average expected movement) and $\varepsilon_t \sim N(0, \sigma^2)$ adds

Gaussian noise, similarly done by Black and Scholes in the 1970s. Mean-reverting behavior can be introduced using an Ornstein-Uhlenbeck-type process with the form $p_{+1} = p + \theta(m - p) + \varepsilon_t$ where m is the long-term mean and $\theta \in (0,1)$ controls the reversion speed. By generating different regimes in separate time windows and stitching them together, we obtain a non-stationary price sequence that mimics the key challenges faced by market-making algorithms.

This approach gives us full control over the structure of the data and enables precise benchmarking of the algorithms' adaptability, regret, and response to regime changes — without relying on any external or historical price feeds.

To summarize we are going to model our data based on:

Market Behavior	**Stochastic Model**	**Price Equation**	**Key Parameter**	**Explanation**
Trending	Random Walk with Drift	$^{t+1} = p + \mu + \varepsilon_t$	$\mu \neq 0, \sigma$	Persistent upward/downward movement. Drift μ drives the trend.
Mean-Reverting	Ornstein–Uhlenbeck Process	$^{t+1} = p + \theta(m - p) + \varepsilon_t$	$\theta \in (0,1)$,m,σ	Pulls price back toward long-term mean m. Used to simulate stabilizing assets.
Volatility	High-Variance Random Walk	$p_{+1} = p + \varepsilon_t$, $\varepsilon_t \sim N(0, \sigma^2)$	$\mu = 0, large\ \sigma$	Unpredictable swings with no clear direction. Resembles news-driven moves.

Pseudocode

Input:

```
T_trend      ,number of time steps for trending regime ,number
T_revert     of time steps for mean-reverting regime
T_volatility ,number of time steps for volatile regime
p_0          ,initial price
μ            ,drift (for trending)
θ            ,mean-reversion speed (for mean-reverting)
m            ,mean price (for mean-reverting)
σ_trend      ,noise std. dev. for trending
σ_revert     ,noise std. dev. for mean-reverting
σ_vol        ,noise std. dev. for volatile
```

Initialize:

```
p_trend[0]   ,p_0
p_revert[0]  ,last value of p_trend
p_vol[0]     ,last value of p_rever


            //Generate Trending Regime//

            for t = 1 to T_trend:
               ε            ,sample from Normal(0, σ_trend^2)
               p_trend[t]   ,p_trend[t−1] + μ + ε


            //Generate Mean-Reverting Regime//

            for t = 1 to T_revert:
               ε                        ,sample from Normal(0,
            σ_revert^2)
               p_revert[t]          ,p_revert[t−1] + θ × (m −
            p_revert[t−1]) + ε


            //Generate Volatile Regime//

            for t = 1 to T_volatility:
               ε            ,sample from Normal(0, σ_vol^2)
               p_vol[t]     ,p_vol[t−1] + ε


            //Concatenate All Regimes//

            p_synthetic ← concatenate(p_trend, p_revert,
            p_vol)

            Output:

               p_synthetic
```

5.Conclusion

This last part of the study, explored how online learning, and in particular the Multiplicative Weights (MMMW) algorithm, can be applied and improved in the prism of market making. While the standard MW algorithm works well in theory, it assumes a stable, stationary environment— something rarely found in real financial markets. To address this, we introduced a simple modification: adding “memory” through discounted learning.

The idea is to reduce the influence of older observations so the algorithm can better adapt to recent and sudden market changes. This concept is inspired by similar ideas in time-series forecasting, reinforcement learning, and bandit algorithms for non-stationary environments. Although related methods exist, i.e the Exponentially Weighted Average forecasters with forgetting factors and some variants of the Hedge algorithm, discounting is not often used in online learning for market making.

To test our approach, we proposed and demonstrated how a synthetic dataset that would simulate different market regimes like trending, mean-reverting, and volatile periods, would be constructed. This would allow us to observe how the algorithm adapts across changing conditions. We also proposed extensions like adaptive discounting and trigger-based memory, which could help make the model even more responsive—though they introduce extra complexity as, again, such changes would drift away from the simple core algorithm and would require a much more financial approach.

In short, we’ve shown that a small change to a classic algorithm can make it more practical for dynamic markets, and that there’s room to build even smarter, more adaptive versions in future research.

Bibliography

1. Abernethy, J., & Kale, S. (2013). Adaptive market making via online learning. Advances in Neural Information Processing Systems, 26.
2. Arora, S., Hazan, E., & Kale, S. (2012). The multiplicative weights update method: a meta-algorithm and applications. Theory of Computing, 8(1), 121-164.
3. Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. Machine Learning, 47, 235-256.
4. Avellaneda, M., & Lee, J. (2010). *Statistical arbitrage in the US equities market*. Quantitative Finance, 10(7), 761–782.
5. Avellaneda, M., & Stoikov, S. (2008). High-frequency trading in a limit order book. Quantitative Finance, 8(3), 217-224.
6. Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In Fourth international workshop on knowledge discovery from data streams (Vol. 6, pp. 77-86).
7. Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In Proceedings of the 2007 SIAM international conference on data mining (pp. 443-448). Society for Industrial and Applied Mathematics.
8. Bouchaud, J. P., et al. (2009). *Handbook of Financial Markets: Dynamics and Evolution*. Elsevier.
9. Bouchaud, J. P., Farmer, J. D., & Lillo, F. (2009). In T. Hens & K. R. Schenk-Hoppe (Eds.), *Handbook of Financial Markets: Dynamics and Evolution*. Elsevier.
10. Branco, P., Torgo, L., & Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. ACM Computing Surveys (CSUR), 49(2), 1-50.
11. Cesa-Bianchi, N., & Lugosi, G. (2006). Prediction, learning, and games. Cambridge University Press.

12. Cesa-Bianchi, N., & Orabona, F. (2021). Online learning algorithms. Annual Review of Statistics and Its Application, 8(1), 165-190.
13. Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D. P., Schapire, R. E., & Warmuth, M. K. (1997). How to use expert advice. Journal of the ACM (JACM), 44(3), 427-485.
14. Chen, L. P. (2019). Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar: Foundations of machine learning: The MIT Press, Cambridge, MA, 2018, 504 pp., CDN $96.53 (hardback), ISBN 9780262039406 [Review of Foundations of machine learning].
15. Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. Journal of Machine Learning Research, 3(Jan), 951-991.
16. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. Journal of Machine Learning Research, 7(Mar), 551-585.
17. Cutkosky, A., & Mehta, H. (2020). Momentum improves normalized sgd. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), Proceedings of the 37th International Conference on Machine Learning (Vol. 119, pp. 2260-2268). PMLR.
18. Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining (pp. 71-80).
19. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.
20. European Parliament and Council of the European Union. (2014). Directive 2014/65/EU of the European Parliament and of the Council of 15 May 2014 on markets in financial instruments and amending Directive 2002/92/EC and Directive 2011/61/EU (MiFID II). Official Journal of the European Union, L 173, 1-148.
21. Fabozzi, F. J., Jones, F. J., & Ferri, M. G. (2021). *Foundations of Financial Markets and Institutions* (5th ed.). Pearson.
22. Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In A. C. P. L. F. de Carvalho & J. Gama (Eds.), Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence (pp. 286-295). Springer Berlin Heidelberg.

23. Gama,J.,Žliobaitė, I., Bifet,A.,Pechenizkiy,M., & Bouchachia, A.(2014). Asurveyon concept drift adaptation. ACM Computing Surveys (CSUR), 46(4), 1-37.
24. Glosten, L. R., & Milgrom, P. R. (1985). Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. Journal of Financial Economics, 14(1), 71-100.
25. Gould, M. D., & Bonart, J. (2016). Queue imbalance as a one-tick-ahead price predictor in a limit order book. Market Microstructure and Liquidity, 2(02), 1650006.
26. Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., & Howison, S. D. (2013). Limit order books. Quantitative Finance, 13(11), 1709-1742.
27. Grossman, S. J., & Miller, M. H. (1988). Liquidity and market structure. the Journal of Finance, 43(3), 617-633.
28. Harris, L. (2003). *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press.
29. Ho, T., & Stoll, H. R. (1981). Optimal dealer pricing under transactions and return uncertainty. Journal of Financial economics, 9(1), 47-73.
30. Hu, C., Pan, W., & Kwok, J. (2009). Accelerated gradient methods for stochastic optimization and online learning. Advances in Neural Information Processing Systems, 22.
31. Hull, J. C. (2017). *Options, Futures, and Other Derivatives* (9th ed.). Pearson.
32. Kearns, M., & Nevmyvaka, Y. (2013). Machine learning for market microstructure and high frequency trading. In D. Easley, M. Lopez de Prado, & M. O'Hara (Eds.), High frequency trading – New realities for traders, markets and regulators (pp. xxx-xxx). Risk Books.
33. Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations.
34. Kivinen, J. (2003). Online learning of linear classifiers. In O. Bousquet, U. von Luxburg, & G. Rätsch (Eds.), Advanced lectures on machine learning: Machine learning summer school 2002 Canberra, Australia, February 11–22, 2002 revised lectures (pp. 235-257). Springer.
35. Kumar, P. (2020). Deep reinforcement learning for market making. In Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (pp. 1892-1894).
36. Lakshminarayanan, B., Roy, D. M., & Teh, Y. W. (2014). Mondrian forests: Efficient online random forests. Advances in Neural Information Processing Systems, 27.

37. Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.
38. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. IEEE Transactions on Knowledge and Data Engineering, 31(12), 2346-2363.
39. McMahan, B., & Streeter, M. (2010). Adaptive bound optimization for online convex optimization. In Conference on Learning Theory (pp. 244).
40. Mishkin, F. S., & Eakins, S. G. (2018). *Financial Markets and Institutions* (9th ed.). Pearson.
41. Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning. MIT Press.
42. Nasdaq. (n.d.). High-frequency trading. Retrieved from https://www.nasdaq.com/glossary/h/high-frequency-trading
43. Nevmyvaka, Y., & Kearns, M. J. (2006). Reinforcement learning for optimized trade execution. In W. W. Cohen & A. W. Moore (Eds.), Proceedings of the 23rd International Conference on Machine Learning (ICML) (pp. 673-680).
44. O'Hara, M. (1995). *Market Microstructure Theory*. Blackwell Publishers.
45. Pérez-Sánchez, B., Fontenla-Romero, O., & Guijarro-Berdiñas, B. (2018). A review of adaptive online learning for artificial neural networks. Artificial Intelligence Review, 49, 281-299.
46. Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In Proceedings of the 24th international conference on machine learning (pp. 807-814).
47. Slivkins, A. (2019). *Introduction to Multi-Armed Bandits*. Foundations and Trends® in Machine Learning, 12(1-2), 1–286.
48. Spooner, T., & Savani, R. (2020). Robust market making via adversarial reinforcement learning. In Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (pp. 2014-2016).
49. Spooner, T., Fearnley, J., Savani, R., & Koukorinis, A. (2018). Market making via reinforcement learning. In Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (pp. 434-442).
50. Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction (Vol. 1, No. 1). MIT Press.

51. Tieleman, T., & Hinton, G. (2012). RMSProp: Divide the gradient by a running average of its recent magnitude. Coursera.
52. Yang, Z., Lei, Y., Wang, P., Yang, T., & Ying, Y. (2021). Simple stochastic and online gradient descent algorithms for pairwise learning. Advances in Neural Information Processing Systems, 34, 20160-20171.