

POLITECNICO DI TORINO

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Matematica



TESI DI LAUREA MAGISTRALE

A reinforcement learning approach to dynamic pricing

Supervisor:
Prof. Paolo Brandimarte

Candidate:
Monica Ferrara

Torino, March 2018

Acknowledgements

First of all, I would like to thank all the waterdata team, for giving me the chance of a job experience that taught me a lot from both professional and relational point of view. Thanks also for passing me the passion for this ambitious project.

In particular, my heartfelt thanks to Simone, for the fundamental and constant support he gave me in doing this work. Thanks even for the time he spent for me and his precious advice. My heartfelt thanks also to Marco, who shared with me the joys and the troubles of programming and data analysis. Thanks to Emanuele too for helping me with technical problems and for his unforgettable suggestions. Many thanks to Federico for his encouragement and thanks to Alessandro for being always available with his concrete help.

My gratitude goes also to prof. Paolo Brandimarte, for his support and his important advice in supervising this work.

Thanks from the heart to my family, for always believing in me in easiest and hardest moments. These few words are not enough to express how big is my gratitude.

My heartfelt thanks to my brother Paolo for his very concrete help and for all the time he generously donated to me. Thanks also to Maria and Federico, who gave me the opportunity of improving my skills in challenging situations.

Many thanks to my friends spread all around the world, for each moment we shared together that gives me the strength of going over difficulties.

Thanks also to you, reader, for being part of my life and for sharing with me this important goal.

*I can do all things
through him who strengthens me.*
— Philippians 4,13

To my nephew...

Abstract

The study of dynamic pricing has grown in the last decades thanks to the scientific advances in statistics, economics and information technology. However, the usual approach to dynamic pricing strategies has some drawbacks in such a way that they are not completely effective. Therefore, there is the need for exploration of new approaches to dynamic pricing. This work was born within an internship collaboration with *waterdata*, a small software company, with the aim of developing an innovative dynamic pricing software, named *Liquidprice*.

Dynamic pricing refers to a class of revenue management strategies that are based on changing prices over time under changing circumstances. These strategies should lead to the achievement of a business goal, through the analysis of the market data. The most difficult step in the design of a strategy is to estimate the sensitivity of customers to different prices. This step should also consider the uncertainty of the market. In order to perform this task, it's possible to use many different machine learning techniques, but the most promising one is the reinforcement learning. It is a goal-directed technique, which is based on learning through interaction between an agent and its environment.

This thesis provides a methodological framework for adapting the reinforcement learning to dynamic pricing problems. In order to study the performances of this pricing algorithm, the software has been applied in the context of flights' insurance.

The result is that the reinforcement learning approach emerges as promising in solving problems that arise in standard approaches. In particular, thanks to their adaptation to real-time data, the reinforcement learning based strategies are able to follow the market changes, therefore taking into account the uncertainty of customers' behaviors.

Sommario

Lo studio del pricing dinamico è cresciuto negli ultimi decenni, grazie ai progressi della scienza in statistica, economia e nella tecnologia dell'informazione. Tuttavia, l'approccio solitamente usato nelle strategie di pricing dinamico porta alcuni svantaggi che le non rendono efficaci. C'è quindi la necessità di esplorare nuovi approcci al pricing dinamico. Questo lavoro nasce all'interno di tirocinio svolto in collaborazione con l'azienda *waterdata Sagl*, con l'obiettivo di sviluppare *Liquidprice*, un innovativo software per il pricing dinamico.

Il termine pricing dinamico si riferisce a una classe di strategie di revenue management, basate sul cambiamento dei prezzi nel tempo al variare delle circostanze. Queste strategie hanno come fine il raggiungimento di obiettivo di business, attraverso l'analisi dei dati di mercato. La fase più difficile nel processo di disegno di una strategia è la stima della sensibilità dei clienti a diversi prezzi. Questa fase deve anche tener conto dell'incertezza del mercato. Per svolgere questo compito sono disponibili numerose tecniche di machine learning, ma più promettente in questo campo è la tecnica di reinforcement learning. Tale tecnica permette l'apprendimento attraverso l'interazione tra un agente e l'ambiente, in modo da raggiungere un obiettivo.

Questa tesi fornisce un impianto metodologico per adattare il reinforcement learning ai problemi di pricing dinamico. Per studiare le prestazioni di questo algoritmo di pricing, il software è stato utilizzato nel contesto di applicazione delle assicurazioni dei biglietti aerei.

Da questo lavoro risulta che il reinforcement learning è una tecnica promettente nel risolvere i problemi che emergono negli approcci più comuni. In particolare, grazie al loro adattamento ai dati ricevuti in tempo reale, le strategie basate sul reinforcement learning sono capaci di seguire i cambiamenti del mercato e quindi di tener conto dell'incertezza nei comportamenti dei clienti. Inoltre, l'uso di *Liquidprice* nel caso studio delle assicurazioni dei biglietti aerei ha portato ad un aumento degli incassi derivanti dall'acquisto delle assicurazioni.

Contents

Acknowledgements	iii
Abstract	vii
Sommario	ix
List of figures	xii
1 Introduction	1
2 Dynamic Pricing	3
2.1 Revenue Management	3
2.1.1 Some history	5
2.1.2 Conceptual framework	6
2.1.3 Generic process	8
2.2 Dynamic Pricing process	9
2.2.1 Models for dynamic price-sensitive demand	10
2.2.2 Common techniques: markdowns, early discounts, and promotions . .	11
2.3 Applications	13
2.3.1 Baseball Game Ticketing	13
2.3.2 Online retail	15
2.3.3 Problems and evolution	16
3 Reinforcement Learning	17
3.1 Definition and main elements	17
3.1.1 Base elements	19
3.1.2 Markov Decision Process	20
3.2 Simple case: multi-armed bandit problem	25
3.2.1 Exploration-exploitation trade-off	26
3.2.2 General solution algorithm	28
3.3 Examples of solution algorithms	30
3.3.1 Exact solution methods	30
3.3.2 Model-based algorithms	32
3.3.3 Model-free algorithms	34
3.4 Bayesian Q-learning	37

Contents

3.4.1	Q-value distributions	37
3.4.2	Action selection	38
3.4.3	Updating Q-values	39
4	Adapting reinforcement learning to dynamic pricing: Liquidprice	41
4.1	General approach	41
4.1.1	Empirical contextual problems	42
4.2	Liquidprice: a software for dynamic pricing	44
4.2.1	Main elements and terminology	45
4.2.2	Application context	46
4.3	Market response models	47
4.3.1	Kernel functions	48
4.3.2	Value distributions	53
4.3.3	Market Models for insurances	55
4.3.4	Market Models in contexts without negative observations	68
4.4	Pricing policy	75
4.4.1	Pricing policies for insurances	75
4.5	Information propagation	83
4.5.1	Wave features space	83
4.5.2	Propagation dynamics	85
5	Conclusions and future developments	89
	References	92

List of Figures

2.1	Demand curve with variable price elasticity.	4
2.2	Three stages demand process for baseball game tickets.	14
3.1	The agent-environment interaction in reinforcement learning problems.	20
3.2	Representation of Markov Decision Process as a sequence of multi-armed bandit problem.	25
3.3	Architecture for the adaptive heuristic critic algorithm.	34
4.1	Integration of Liquidprice with merchant's selling systems.	44
4.2	Modules interactions for Liquidprice functionality.	45
4.3	Temporal decay coefficient for different values of the halving time parameter h	48
4.4	Gaussian kernel function with $m = 0$ and $\sigma = 1$	49
4.5	Sigmoid kernel function with $m = 0$, $b = 1$ and $c = 0.99$	50
4.6	Sigmoid kernel function with $m = 0$, $b = 1$ and $c = 0.11$	51
4.7	Sigmoid Higher Floor kernel function with $m = 0$, $b = 1$, $c = 0.99$ and $f = 0.1$	51
4.8	Sigmoid Higher Floor kernel function with $m = 0$, $b = -1$, $c = 0.99$ and $f = 0.5$	52
4.9	Left hat kernel function with $m = 0$, $a = 0.8$, $b = 3$, $\sigma_1 = 1$ and $\sigma_2 = 1$	52
4.10	Estimate and Precision curve for Unit Margin Market Model - case 1.	57
4.11	Estimate and Precision curve for Unit Margin Market Model - case 2.	58
4.12	Positive and Negative estimator for Attach Rate Market Model with Sigmoid kernel function - case 1.	61
4.13	Estimate and Precision curve for Attach Rate Market Model with Sigmoid kernel function - case 1.	62
4.14	Positive and Negative estimator for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 1.	64
4.15	Estimate and Precision curve for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 1.	65
4.16	Positive and Negative estimator for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 2.	66
4.17	Estimate and Precision curve for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 2.	67
4.18	Addition of dummy market events for historical data.	68
4.19	Addition of dummy market events.	69

List of Figures

4.20 Historical dummy events - comparison between estimate curves.	70
4.21 Historical dummy events - comparison between precision curves.	71
4.22 Dummy events - comparison between estimate curves.	73
4.23 Dummy events - comparison between precision curves.	74
4.24 Estimate curve and Precision curve for a Unit Margin Market Model.	77
4.25 Precision curve for an Attach Rate Market Model.	78
4.26 Comparison between Estimate curve and Expected Margin curve for an Attach Rate Market Model.	78
4.27 Objective function for the Upper Confidence Bound Policy based on a Unit Margin Market Model.	80
4.28 Objective function for the Lower Confidence Bound Policy based on a Unit Margin Market Model.	81
4.29 Transformation process for determining the dimensions of the Wave features space.	84
4.30 Different choices of γ in the function that transform distance into weight. . . .	86
4.31 Wave dynamics: propagation allowed by overall distance.	86
4.32 Wave dynamics: no-propagation due to blocking dimension.	87

1 Introduction

The scientific advances of the last decades have led to an explosion of the information and data available to companies for evaluating their business. In particular, research in information technology has given the opportunity of increasing the quantity of data to be analyzed in order to take business decisions.

This large amount of data lays the foundations for a new era of revenue management. In fact, many companies found the necessity of performing a more effective data analysis in order to help managers in taking business decisions. These decisions are now really data-driven since they are based on what really happens in the world instead of the human interpretation of what happens.

A field of revenue management that recently rises is called dynamic pricing. It consists of finding strategies for changing the price over time of a selling product based on changing circumstances. In practice, the best strategy should choose the right price for the right customer at the right moment. Clearly, the real market is very uncertain since customers behave in different ways, based on their own sensibility, when they want to purchase a product.

Many companies have tried different dynamic pricing strategies, based on an exploration of customers' demand of a product. However, not all of them are effective when the market is highly uncertain and changes over time. The aim of this work is to design dynamic pricing strategies that are able to take into account this uncertainty.

In order to achieve this goal, this thesis was developed during an internship collaboration with *waterdata*, a young Swiss company. Its main product, in fact, is a software called *Liquid-price* that can be used by companies as a platform for implementing dynamic pricing. Hence, this work presents the methodological framework implemented by Liquidprice for designing dynamic pricing strategies in different application contexts.

The most challenging step in defining a strategy is the estimate of customer behavior on purchasing products in uncertain markets. In order to do this in an effective way, this work will explore a family of recent machine learning techniques, called reinforcement learning. It is a goal-directed learning technique based on the interaction between an agent and its environment. This interaction is the most important element that leads to an effective exploration of customers demand and it could take into account the uncertainty of the market. Therefore, through this technique, it's possible to design dynamic pricing strategies based on the real behavior of customers when purchasing a product.

The use of reinforcement learning for dynamic pricing is not yet common. Therefore, it represents the most important contribution of this work to the existing literature of both dynamic pricing and reinforcement learning.

This thesis is divided into three parts. The first one presents dynamic pricing in relation to revenue management and its history. Moreover, it will provide the general process used for defining a pricing strategy and some examples of applications. These examples will be used to present the main challenges that a common approach has to face.

The second part is devoted to studying the reinforcement learning technique in all of its components and theoretical framework. Moreover, since there are many possible algorithms for this goal-directed learning, a survey of the algorithms that represent the foundation idea in the application to dynamic pricing is presented.

The last part represents the center of this work. First, it will describe how, in general, reinforcement learning can be used for dynamic pricing. Then, it will present the pricing algorithm implemented by Liquidprice. In order to see in practice how this algorithm works, the methodological description is enriched by its application in the context of pricing flight's insurance.

2 Dynamic Pricing

This chapter will introduce the main subject of this work: dynamic pricing technique. It will present the history of dynamic pricing and its relations with the revenue management. Moreover, using some examples of application, it will explore the main challenges that companies have to face in order to implement it.

2.1 Revenue Management

The expression "dynamic pricing" is often used as a synonymous with "revenue management" or "yield management", but this is not completely correct. Indeed, revenue management refers to the "collection of strategies and tactics that firms use to scientifically manage demand for their products and services" [13], while dynamic pricing is only a class of these strategies.

Every seller of a product or a service has faced, at least one time in his life, a number of fundamental demand-management decisions, such as structural decisions about market situations (like exploring and exploiting the consumers' willingness to pay), price decisions and the opportunity of making possible discounts or promotions, quantity decisions related to supply chain management (like control inventory capacity). Depending on the business, these decisions may be taken very frequently and revenue management is a useful support for every manager.

An important concept in these decisions is the price elasticity, which, in many cases, is the main foundation of revenue management strategies. The price elasticity of demand is defined as "the relative change in demand produced by a relative change in price" [13] and it is used to measure how changes in price affect demand changes.

Denoting the price by p and the corresponding demand at that price by $d(p)$, price elasticity can be written as

$$\epsilon(p) = \frac{\Delta\% d(p)}{\Delta\% p} = \frac{\partial d(p) / d(p)}{\partial p / p}$$

Chapter 2. Dynamic Pricing

From this definition, it's possible to distinguish two different demand behavior, depicted in figure 2.1:

- $|\epsilon(p)| > 1$: demand is said to be elastic, which means that small variations in price imply large variations on demand. If $|\epsilon(p)| = \infty$ demand is perfectly elastic.
- $|\epsilon(p)| < 1$: demand is said to be inelastic, that is large variations in price imply small variations in demand. If $|\epsilon(p)| = 0$ demand is perfectly inelastic.

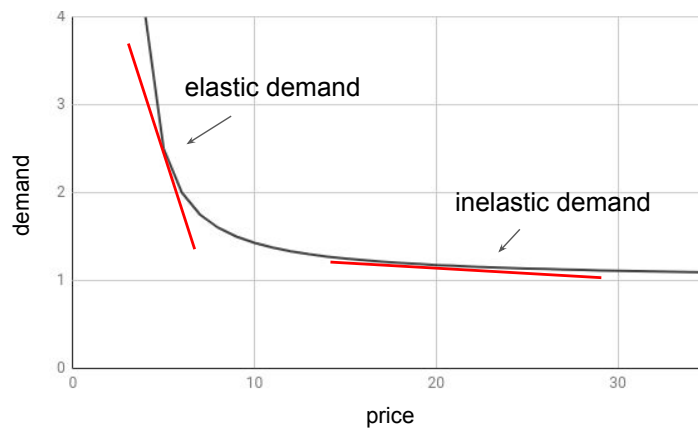


Figure 2.1: Demand curve with variable price elasticity.

Of course, this definition of elasticity is just an estimate of the real one, since it depends also on other factors. Hence, it's important to study elasticity in relation with other factors affecting demand with efficient revenue management strategies.

Beside the concept of elasticity, there is the concept of price-based market segmentation. It assumes that for a product that is on sale, each customer has a willingness to pay, that is the maximum price at which he would buy that product. Hence, if the price of that product is higher than his willingness to pay, he would not buy that good. Otherwise, if the price is lower than his willingness to pay, the customer would buy the product and he would ideally profit of a gain since he has bought something at a lower price than he would expect. This induces a natural segmentation of the market: customers are subdivided into groups based on their willingness to pay.

Therefore, a good revenue management strategy should consider this segmentation in order to exploit the willingness to pay of each customer. In fact, proposing to all customers the same price could be the cause of revenue losses for companies: a low price would be appreciated by all customers with an higher willingness to pay, but it would lead to less potential profits coming from these groups of customers. On the other hand, a high price would be disregarded by customers with a lower willingness to pay, leading to a less number of product sold.

Moreover, customers with different willingness to pay appear in the market in different moments. This implies that the customer segmentation changes over time. Then, revenue management strategies should be able to take into account also this price segmentation over time, in order to propose the right price at the right moment to the right customer.

The idea of revenue management is nothing really new since many people in the past have always tried to solve these problems. What is new is how demand-management decisions are made: a technological and operational method of decision making.

This new approach has been possible thanks to scientific advances in economics, statistics, and operations research and to information technology advances such as big data analysis. Hence, the combination of science and technology with the human interaction makes it possible to manage demand on large scale and to improve the quality of management decisions.

2.1.1 Some history

Revenue management comes from the airline industry. Until 1978, airline prices were strictly regulated by the U.S. Civil Aviation Board with standardized prices and profitability targets, but after the publication of the Airline Deregulation Act, the U.S. CAB loosened their control and the established carriers were free to change prices, schedules, and service: this was the first step towards revenue management.

The industry changed rapidly: large airlines accelerated the development of their systems of computerized reservation and global distribution, while new low-cost and charter airlines entered the market, being able to price much lower than the major airlines. With prices sufficiently low, many people started using planes instead of cars and demand had a strong increase. It also turned out that air travel had a demand quite price elastic [13].

However, although the rapid rise of low-cost carriers as PeopleExpress, large airlines had more schedules and better services that were more important than price for business travelers. Nevertheless, large airlines had too many losses in revenue so that they needed a strategy to recapture the leisure segment of the market.

The first solution to this problem was proposed by Robert Crandall from American Airline: observing that most of the cost of a flight is fixed and that there are many seats at a marginal cost near to zero, he thought that American Airline could afford to compete with the upstarts using these surplus seats. However, there were still two problems in order to execute this strategy: they should have some way of identifying the surplus seats on each flight and they had to be sure that American's business customers did not buy the new low-price products.

American Airlines solved these problems with a combination of purchase restrictions, directed in particular to business travelers (discounts were applied only 30 days in advance of departure, were nonrefundable and required a seven-day minimum stay) and capacity-

controlled fares (the number of discount seats sold on each flight was limited). This combination, called American Super-Saver Fares, helped to compete with the new low-cost airlines, but after the initial success, there were some problems implementing this strategy.

In the first implementation of the strategy, capacity controls were based on applying the new low-fare products to a fixed portion of seats on each flight, but after some time American realized that not all flights had the same demand pattern. Hence, they needed a more intelligent approach to realize the full potential of capacity-controlled discounts: the Dynamic Inventory Allocation and Maintenance Optimizer system (DINAMO), fully implemented in 1985. DINAMO made American able to announce low fares on many different flights, with different capacity control depending on each individual departure and it could match the rival airlines' fares. The effect of DINAMO on the new upstarts was dramatic: PeopleExpress, for example, failed soon, since American repeatedly matched their prices in every market.

After DINAMO system, revenue management in the airline industry was developed more and more and it's now pervasive and mature, such that now any modern airline could not be profitable without revenue management.

However, this strong connection between revenue management and airline industry has two negative faces: there's a sort of myopia inside the field of revenue management since it's considered usable only for airlines and there's a bad reputation of airline pricing among customers, since they don't like to pay different prices for the same flight. Hence many managers are sometimes reluctant to try revenue management in their own business; yet the reality is that, in many cases, applying new pricing strategies is just a way of making more intelligent decisions and does not involve changing the structure of pricing and sales practices.

2.1.2 Conceptual framework

Despite the airline context, revenue management applies in any business where tactical demand management is needed and there are sufficient technology instruments to implement it. In order to find some practical business, it's important to design a conceptual framework for studying the demand management process.

First of all, it's notable to observe that any firm's demand has three natural dimensions: different products, types of customers and their behaviors, and time. Hence, revenue management addresses "the structural, price, timing, and quantity decisions that a firm has to make in trying to exploit the potential of the multidimensional demand" [13].

For example, some revenue management problems exploit the customer heterogeneity by fixing product and time dimension and optimizing over the customer dimension (such as auctions - [13] chap.6); other problems try to dynamically price a single product to different customers over time by fixing product dimension and optimizing over the other two (as dynamic pricing problems); others, finally, manage decisions for multiple products over multiple time periods and do not consider explicitly the customer dimension (such as network prob-

lems - [13] chap.3). In practice, all of the three dimensions are important, but sometimes, for methodological reasons, it's useful to simplify the problem by decomposing the dimensions.

Moreover, there are one or more of the following three factors that link the demand decisions across these three dimensions. First, multiple products may share production capacity or have joint production costs, then products decisions in different periods are interrelated. Moreover, customers may choose among substitute products at any time or may schedule different purchasing so that price or quantity decisions of the firm may affect actual or future demand. Finally, demand decisions for different products, customers, and time-period may be linked over time in terms of information the firm gains, thus a price decision now may affect information about demand sensitivity. Hence, it will affect future pricing decisions. These links make demand management decisions more complex but lead to a stronger motivation about the necessity of revenue management.

Given this framework, it's possible to have insights into business conditions where revenue management could make the difference:

- Customer heterogeneity: if customers are heterogeneous, then it's possible to exploit their variations in willingness to pay, preferences for different products and variations of purchase behavior in order to take strategical decisions that could improve revenues.
- Demand variability and uncertainty: if demand varies over time, due for example to seasonality or shocks, it's hard to estimate future demand since there's a lot of uncertainty. Hence, it's useful to have sophisticated tools for making management decisions that take uncertainty into account.
- Production inflexibility: if the production is inflexible, it's difficult to match demand variations with supply variations; then, accurate demand management decisions are important.
- Price not as a signal of quality: often, the price of a product is a signal of quality to customers, but in these contexts, revenue management does not suite well, since quality signals are subjective and not shared among different customers.
- Information system infrastructure: revenue management is more effective in industries that use databases and transaction processing systems for collecting and storing the data needed to characterize and model demand.

It's easy to see that the airline industry satisfies all these conditions. Other industries adopting revenue management technology are, for example, service industries (as hotels, cruise ship lines, car rental companies, theaters and sporting venues), retailers (especially fashion apparel, consumer electronics and toy sector), as well as the energy sector.

2.1.3 Generic process

Generically, any revenue management system follows a standard process, which can be summarized in the following four steps:

- Data collection: collect and store relevant historical data, such as prices, demand, and causal factors.
- Estimation and forecasting: first, estimate the demand model and its parameters, then forecast future demand and other relevant quantities related to management decision to be taken, based on transaction data.
- Optimization: find the optimal set of controls, such as allocations, prices, markdowns, discounts, to apply until next re-optimization;
- Control: control the sale of inventory using optimized control.

Depending on the decisions frequency, these steps can be cyclically repeated at determined intervals. Each repetition can involve different types of controls, that are strongly related to the specific decisions that are to be taken. In particular, primary demand management decisions may concern two main scopes, hence revenue management processes can be divided into two categories: quantity-based or price-based revenue management [13].

In quantity-based revenue management, the main decisions concern product rationing and availability control, as how much to sell to whom, whether to accept or reject requests for products and so on. It includes problems like Single-Resource Capacity Control, Network Capacity Control, and Overbooking that are typical of hotels and rental car industries.

In price-based revenue management, instead, the primary decisions are prices, such as how to price to various customer groups or how to vary prices over time. It includes posted prices and auction mechanisms that are used in retail and manufacturing industries.

However, these two classes of strategies are not really separated, since many industries employ a combination of both techniques in their revenue management practices. Moreover, in many situations, price-based revenue management has an impact also on control quantity, then it's the preferred choice.

2.2 Dynamic Pricing process

Dynamic pricing is then the main strategy for price-based revenue management and concerns studying techniques that explicitly use price as control variable and model demand as a price-dependent process. In other words, dynamic pricing is "the study of determining optimal prices under changing circumstances", where changing circumstances can be changes in the market, competitors decisions, innovations techniques, changes in consumer tastes, changes in inventory levels but also other external conditions like weather forecasts [4].

Moreover, it turns out that determining the right price for a product is a complex task since it requires that a company should know operating cost and inventory availability, and also how much the customer values the product and the future demand [5].

However, dynamic pricing is nowadays widely applied in many contexts: airlines, trains, car rental, accommodation, ticketing (concerts, theaters, amusement parks), retail and e-commerce. Firms usually use different ways of varying prices, including personalized prices, markdowns, promotions, coupons, discounts, sales, auctions and price negotiations, in order to respond to market fluctuations and uncertainty in demand.

The dynamic pricing process can be easily derived from the process for a generic revenue management system with some advice: the data collection is focused on capture price-sensitivity of customers in relation to products and obviously using the price as control variable. The final output of the process is the optimal pricing strategy: a rule or an algorithm that explains what is the best price to be applied at each moment of the selling period. The form of this strategy depends on the business considered and should be investigated in every context.

Dynamic pricing problems are classified depending on the number of product considered: single-product problems consider the pricing of a single item, without considering the relationship with other items; while multi-product problems aim to price many products together, considering mutual connections between items (such as correlated demand and capacity constraints). These two scenarios have an important impact on the model definition and on its complexity.

Another classification is given by the type of inventory: some industries have an inventory that can be replenished in any time period at a certain cost, while other firms have a fixed inventory. This distinction plays an important role in the strategy design since in the first case pricing decisions must be coordinated with supply decisions.

2.2.1 Models for dynamic price-sensitive demand

A fundamental step in the dynamic pricing process is the design of a model describing how demand responds to changes in price. The study of demand models is wide and complex and it's beyond the scope of this work, but it's important to highlight some important assumptions for these models, concerning how individual customers behave over time and the state of market conditions.

Some models consider myopic customers: they buy as soon as the offered price is less than their willingness to pay. Meanwhile other models consider strategic customers who will optimize their purchase behavior in response to pricing strategies of the firms. Strategic customers are more realistic, but a model with myopic customers is more tractable and hence is most widely used. This assumption is also justified by this fact: customers are sufficiently spontaneous in making decisions in such a way that one can ignore their strategic behavior. Moreover, customers often do not have sufficient time or information to behave strategically.

Another assumption is whether the population of potential customers is finite or infinite. In reality, the population of potential customers is finite, since it's reasonable to consider that the number of customer's types is finite. However, some models assume an infinite population, that means a sampling with replacement among the observed customers. This implies that future demand is not affected by the past observed demand. This assumption is called nondurable-goods assumption since bought product is immediately consumed by the customer and hence he reenters in the population of customers.

Conversely, the finite-population models assume a sampling without replenishment, which is called durable-goods assumption, since, after a purchase, the customer is removed from the population for a long period of time in which he consumed the good. In this case, population is composed of a finite number of possible customers with different willingness to pay values and past pricing decisions need to be tracked, since they affect future decisions of residual customers.

For example, if a product is offered at a certain price p , then all customers (with myopic behavior) who value that item at more than p will purchase that product. Hence, after the period of that offering price, the willingness to pay of the remaining customers will be almost surely less than p , i.e. it's influenced by past demand.

Although the infinite-population model is analytically easier, the choice between the two models depends on the context, especially on the number of potential customers relative to the number that actually buy and on the type of good. Anyway, the two models lead to different pricing policies: finite-population models have price skimming as optimal price, which means lowering prices over time aiming to sell early at higher prices to high-valuation customers and sell in later periods at lower prices to low-valuation customers. This strategy does not apply to the infinite-population model, since in any time periods there are customers with different price valuations, so it's not profitable to lower prices over time.

The last assumption is about the level of competition the firm faces: monopoly, oligopoly or perfect-competition. The monopoly models assume that firm's demand depends only on its own price and not on the price of its competitors, hence it does not take explicitly into account the competitive reaction to a price change. Although this assumption may lead to distorted vision of demand since in reality competition is an important feature to be considered, these models are considered valuable for decision support.

However, others prefer using oligopoly models, in which "equilibrium price response of competitors is explicitly modeled and computed" [13]. These models are useful to have insights on issues of pricing strategies, even though this is a too strong approximation to be applied in practice. Finally, in perfectly competitive models the assumption is that many competing firms supply an identical commodity. As a result, with this assumption, firms cannot influence market prices, which means that they sell only at the prevailing market price. Hence, this model is not so useful in price-based revenue management, but it plays an important role in quantity-based revenue management.

2.2.2 Common techniques: markdowns, early discounts, and promotions

In many industries, the selling goods are perishable in a short time or they have low salvage values when the sales season is over. In these cases, the strategy used by the firms is a price reduction, called markdown pricing in order to clear excess inventory before the end of the season. Indeed, firms prefer selling inventory while they can, even at a low price, rather than salvage it. This technique is used in industries as apparel, sporting goods, high-tech, and perishable-foods retailing.

Markdown pricing is even used as a form of demand learning. In fact, retailers are often uncertain about which products would be popular with customers, then all items have high initial price. Customers will buy popular products before the others, even if they have high prices; on the other hand, the remaining items are marked down and sold at a lower price. In this way, firms can learn which types of products are most popular among customers and which not.

Markdown pricing serves also as a segmentation mechanism to separate price-insensitive customers from those price-sensitive customers that want to defer consumption to get a lower price. Actually, customers who purchase early have higher willingness to pay, either because they can use the product for a full season or because they would like to have the exclusive on that particular product. Moreover, experience shows that this customer behavior changes on holidays and during peak-shopping periods. In these situations, customers spend more time in looking for the best price than what they would have used in a normal period. This results in a more price-sensitivity demand, then retailers are more involved in making discounts during these periods.

Chapter 2. Dynamic Pricing

However, in other industries, price reductions at the end of the selling period is not as profitable as in the retail industry. For example, airlines vary ticket prices based on capacity and demand for a specific departure: usually, they offer only one type of ticket on each flight (a non-refundable, one-way fare without advance-purchase restrictions) at increasing price during the selling period.

This increasing pattern is motivated by the fact that, differently from the seasonal-goods products, the value of air travel to customers is typically low for early purchase (since they're not sure of really using the ticket) and this value does not necessarily go down as the deadline approaches. In fact, demand is less price-sensitive close to the time of service; then in the airlines, discounts are made in the early selling period.

In the consumer packaged-goods industry, the most common dynamic pricing technique is the promotion, which is a short-run and temporary price reduction. In this industry, products are sold repeatedly and customers are often aware of past prices. Hence, a brand that runs frequent promotions has a positive impact to customers in their subjective evaluation of the fair price.

2.3 Applications

This section will present two examples of an application of the dynamic pricing process in different business contexts. These are useful to explore possible ways of finding pricing strategies and to find possible improvements.

2.3.1 Baseball Game Ticketing

The first example of dynamic pricing strategy comes from sports ticketing. Three researchers from USA (Xu, Fader and Veeraraghavan [17]) observed that often managers are not able to evaluate the revenue impact of dynamic pricing and to model properly the demand process, hence they studied these issues in the context of a selling season of single-game tickets of a Major League Baseball franchise.

Business managers of this franchise introduced dynamic pricing for tickets of home games during the second half of the season, while in the first half they used a fixed pricing policy, in which ticket prices varied across the 81 games and 14 seat sections but remained constant over time. The dynamic pricing strategy allowed prices to vary over time and was updated daily in order to maximize single-game ticket revenue while respecting certain internal business rules and constraints.

Dynamic prices were determined by a committee: a third-party software provided price recommendations based on remaining selling horizon and inventory, then the prices were manually updated with the incorporation of additional information such as team performance forecasts. Moreover, these prices had to respect two internal pricing rule: the first rule concerned the price relationship among seat sections (connection between price and quality of seats), while the second specified that ticket prices could not decrease over time.

Since the franchise was not able to perfectly control price experiments and their impact on revenue, the three researchers proposed a model to evaluate the effect of dynamic pricing and a new optimized strategy to improve revenue.

Modeling price-sensitivity of customers in this framework is not so easy since there are some involved factors that are not quantifiable. However, the ability to generate revenue forecasts given a specific price path is an integral component of dynamic pricing.

The first step to this goal was to develop a demand model, using stochastic processes, that took into account the three stages of the ticket purchase process: game demand, ticket quantity choice, seat section choice. The ticket purchase demand process is depicted in figure 2.2: at first, the client could choose between the m games on sales, then he choose the quantity of tickets to be purchased and finally the seat sections.

The model variables used for building this model considered time effects, game characteristics, team performance, price effects and popularity effects.

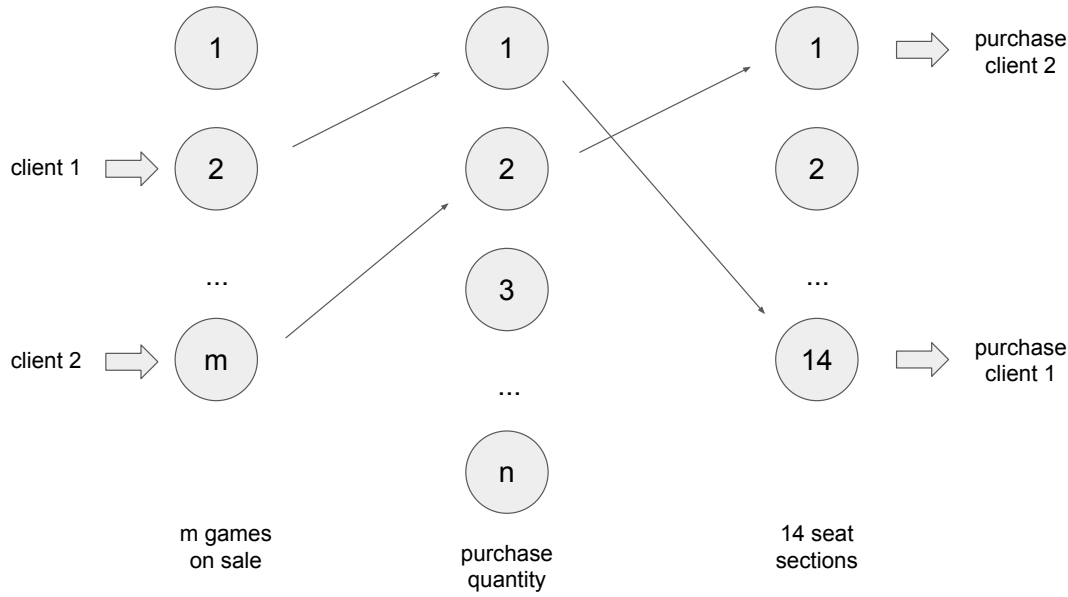


Figure 2.2: Three stages demand process for baseball game tickets.

In particular, the number of daily orders (game demand) was modeled by a negative binomial regression, in order to capture overdispersed count data. For the same reason, the ticket quantity choice was modeled with another negative binomial regression, independent from the previous one. This second stage of the model directly affected the seat section choice which was represented by a standard multinomial logit model, useful to consider the trend of purchasing the more expensive seat sections earlier than others and the fact that an order with more tickets is less likely to purchase more expensive seat sections.

Since the season data were divided into two parts based on the pricing policy of the franchise, this three-stages demand model was estimated with data of the first half of the season (fixed price), while the second half of the season was used to test the model fitted. After this validation, the output of the demand model was then used to compute the expected revenue given a price path.

Moreover, this model was used to observe many factors affecting demand and price-sensitivity (such as team performance), leading to more precise and effective dynamic pricing policies. Indeed, they observed that the franchise policy led to a slight revenue loss (-0.5%) with respect to the fixed price policy. Hence, an optimized implementation could perform better.

The policy proposed by the researchers was a daily re-optimized pricing policy: seat section prices for each game were optimized on a daily basis to maximize the expected daily revenue and the remaining inventories were adjusted according to the expected sales from the previous day's optimization result. This policy could then account for changes in exogenous market

conditions in addition to remaining inventory and selling horizon and it turned out that it can improve revenue by 11.4% over the initial fixed pricing policy.

It's notable to observe that this policy allowed price decreasing over time (even if always above the initial price), giving an empirical proof that increasing prices is not always an optimal strategy to improve revenue. Indeed, a daily re-optimized policy without allowing price decreasing from previous day's price resulted to improve revenue only by 3.8%. Moreover, these results could be improved up to 17.2% allowing a maximum of 5% discount from the initial price.

In conclusion, this example of multiproduct dynamic pricing problem is useful to observe that a well-calibrated demand model can help to implement and evaluate a more effective pricing policy.

2.3.2 Online retail

Another example concerns an online retailer, who wants to optimize its prices in order to maximize revenue (Ferreira, Lee, Simchi-Levi [6]). The retailer is an online fashion sample sales industry, that offers extremely limited-time discounts on designer apparel and accessories.

Products' catalog showed on the website is organized in "events", representing a collection of products that are similar in some way. Each event has a limited available time and after it, remaining inventory is planned to be sold in a subsequent event. Products sold for the first time are called "first exposure styles" and they are the most profitable products, but it may happen that inventory sold out before the end of the selling period or that there are too many unsold products at the end of the period. Hence the management would like to investigate demand and pricing policies in order to maximize revenue from first exposure sales.

The first step is to analyze past data in order to study the demand for past products. In particular, they estimate the percent of sales occurred each hour of the available time of each event, obtaining an empirical distribution of the proportion of sales occurred in a period. This process results in a lot of distributions, so they use a clustering algorithm to aggregate similar distributions. Hence, they're able to estimate demand for every product showed in the past, also for products that were sold out in a certain hour.

Demand estimates within features of historical data are then used to build a regression model able to predict demand for future first exposure styles. They try many models and it turns out that the best are regression trees with bagging, that are very interpretable models.

The last step consists of determining an optimal pricing strategy that maximizes revenue, given predetermined purchasing and assortment decisions, since the retailer cannot buy inventory based on expected demand. Practically, they build an optimization problem where the objective is to select a price from a given set of prices for each style, in order to maximize the revenue earned from these styles in their first exposure.

This example presents a very simple dynamic pricing strategy, but it has an interesting challenge in demand modeling since there is uncertainty about future demand and there are missing data about sold out products.

The problem of missing data is very common in statistical analysis when there is no data value stored for a variable in an observation. In this case, missing data comes from missing observations about possible purchases on products that are not available anymore. There are many techniques to face up with this problem and this example gives an idea of a possible solution.

2.3.3 Problems and evolution

As seen in the previous examples, a difficult task in dynamic pricing problems is modeling price sensitivity of customers, since there's always uncertainty about consumers' response to different selling prices. Moreover, in many contexts, the large amount of available sales data are not efficiently used to exploit market trends or price-sensitivity [4].

Then, dynamic pricing problems concern not only price optimization, but it's also about learning the relationship between price and market response. This connection can be modeled by estimating a demand model from historical data but could be enforced with online learning, which means updating the model every time new data are collected. Thus, one important task in solving dynamic pricing problems is to find a way of conducting simultaneously estimation of price-demand relation and optimization of selling prices.

These class of problems is called sequential decision problems under uncertainty: a decision maker has to choose an action in order to reach an objective, without knowing the optimal one. After trying different actions, the decision maker can observe the different outcomes and learn the quality of each action. But now, he could select the best action from his knowledge, or select another action to improve his knowledge.

A possible approach to solve these problems is reinforcement learning, which will be presented in the following chapter.

3 Reinforcement Learning

This chapter will explore a computational approach to goal-directed learning, known as reinforcement learning. This is an approach which learns from interaction with the external contexts and it is very useful for implementing dynamic pricing methodologies. Reinforcement learning will be explored in both theoretical elements and solution algorithms.

3.1 Definition and main elements

Reinforcement learning is "the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment" [8]. Practically, these problems concern learning what to do (i.e. how to map situations to actions) so as to maximize a numerical reward signal [12].

The reinforcement learning problems have three important distinguishing features. First of all, they could be seen as closed-loop problems since actions of the learning system influence its later inputs. The second characteristic is that the learner does not know what action has to take, but has to discover which is the best, by trying them out. Finally, actions have consequences also on all subsequent reward, not only to immediate reward.

The full definition of the reinforcement learning problem will be given below by Markov decision process and its idea is to capture aspects of the real problem by observing a learning agent interacting with the environment to achieve a goal. Moreover, the agent has to sense the environment's state in order to select actions affecting state and reaching his goal. Any method able to solve this kind of problems is considered a reinforcement learning method.

Note that reinforcement learning is different from supervised and unsupervised learning. In supervised learning, the agent has to learn how to act from labeled examples provided by an external supervisor and generalize them to other situations. Since reinforcement learning is an interactive problem, it's often impossible to obtain examples that are representative of all possible situations that the agent can face up with. Then, the interactive agent has to learn from its own experience. On the other hand, unsupervised learning concerns finding structure

hidden in collections of unlabeled data, and this seems to be similar to reinforcement learning, but the goal of interactive systems is to maximize a reward signal instead of finding hidden structures.

In addition, the main challenge that distinguishes reinforcement learning from other kinds of learning is to find the trade-off between exploration and exploitation: the agent should exploit what he already knows in order to obtain a reward, but also he should explore in order to make a better action selection in the future. The challenge is not in performing each of these tasks, but to perform them simultaneously in order to reach the goal.

Some examples of a reinforcement learning problem could be the following:

- Chess game: in order to win the game, the player has to learn which is the best move to make in any situation he can be after the opponent's move.
- Robot collecting cans: the robot has to decide whether to enter a new room in search of more cans to collect or to go back to the battery recharging station. Its decision is based on the current charge level and if it's quick to find the recharger, based on past experience.
- Preparing breakfast: any person who wants to prepare his breakfast in the faster way has to learn the best sequence of moves to do, after observing the actual kitchen's situation and where the objects are located. Moreover, he has to learn what is the best time for warming milk or for toasting bread, based on his preferences.

All of these scenarios involve interaction between an active decision-making agent and its environment, within which the agent seeks to achieve a goal despite uncertainty about its environment. The agent's actions could affect the future state of the environment and their effects cannot be fully predicted, hence the environment must be monitored frequently by the agent who should react to any situation. The knowledge of the environment's state is also useful to improve performances over time.

Reinforcement learning problems have grown fast in the last years and have found interactions with other engineering and scientific disciplines as well as applications in psychology and neuroscience.¹

¹ for detailed history of reinforcement learning, see [12].

3.1.1 Base elements

As seen, in any reinforcement learning system, the agent and the environment play an important role, since they are the primary actors interacting in the system. Beyond them, there are four more elements to completely describe the system:

- **Policy:** it's a, possibly stochastic, mapping from perceived states of the environment to actions to be taken in those states, defining the learning agent's way of behaving at a given time. The policy is the core of the learning agent since it's sufficient to determine behavior.
- **Reward signal:** it indicates what are good or bad events for the agent in order to reach his goal. In practice, at each time step, the agent receives from the environment a single number, as an immediate reward for his action. Then, the objective of the agent is to maximize the total reward over the long run. The agent cannot alter the process that generates reward, but he can change the signal he receives by changing the selected action. This means that reward signal is the primary element for modifying the policy. In general, reward signals may be modeled as stochastic functions, depending on the environment's state and on the actions taken.
- **Value function:** it specifies what is good in the long run, in the sense that it indicates the long-term desirability of states, after taking into account states that are likely to follow and their reward. Practically, the value of a state is the total amount of reward that an agent can accumulate over the future, starting from that state.
- **Model of the environment:** something allowing inference on how the environment will behave. Models are used for planning, which means finding a way of deciding an action considering known situations before being experienced. Models are not always necessary for solution methods in reinforcement learning problems: if they are used, solution methods are called model-based methods, otherwise model-free methods.

Many solution algorithms consider the value function as the most important element since it's the one used by the agent when he has to make and evaluate decisions: action choices are made based on the value judgment. Unfortunately, the value function is not easy to estimate. In fact, while rewards are given directly by the environment, values must be estimated from the sequence of observations an agent makes over its lifetime. Hence, a method for efficiently estimating values has a central role in most of the reinforcement learning solution method.

3.1.2 Markov Decision Process

This section will present in a more formal way the generic problem that any reinforcement learning method should try to solve, based on [12]. The idea of the problem is learning from interaction to achieve a goal. Interaction is between the agent, who is the learner and decision maker, and the environment, which is everything outside the agent. In practice, the agent selects actions and the environment responds to those actions (in the form of reward, that the agent tries to maximize over time) and presents new situations to the agent.

Without loss of generality, it's possible to assume that interaction takes place at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \dots$, at which the following elements are defined:

- the environment's state $S_t \in \mathcal{S}$, where \mathcal{S} is the set of all possible states. A state could be anything that is useful to characterize the environment in order to take actions.
- the set $\mathcal{A}(S_t)$ of actions available in state S_t . An action $A_t \in \mathcal{A}(S_t)$ could be any decision the agent wants to learn how to make.
- the numerical reward $R_t \in \mathcal{R} \subset \mathbb{R}$ due to the action that was taken in the previous time step.
- the mapping from states to probabilities of selecting each possible action, implemented by the agent, called policy π_t where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

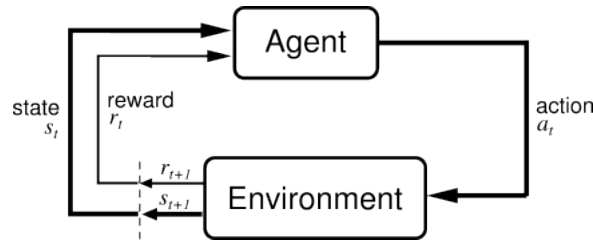


Figure 3.1: The agent-environment interaction in reinforcement learning problems.

As visualized in figure 3.1, at each time step t , the agent observes his current state S_t and on that basis, it selects an action A_t , following his policy. In the subsequent time step, the agent will receive the reward R_{t+1} from the environment and will find itself in a new state S_{t+1} .

In general, the environment could be deterministic or non-deterministic (taking the same action in the same state on two different time steps may or not result in different next status and/or different reward) and non-stationary or stationary (probabilities of making state transitions or receiving specific rewards may or not change over time) [8].

In this framework, any reinforcement learning solution method specifies how the agent changes his policy as a result of his experience, knowing that his goal is to maximize the total amount of reward received over the long run.

One of the distinctive features of reinforcement learning is the use of reward signal to formalize the goal that is stated by the reward hypothesis:

«All of what is considered to be a goal or a purpose can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (the reward)» [12].

The computation of reward signal is defined in the environment and depends on the problem application: the agent cannot modify rewards' generation, but he can only observe their realization after selecting an action.

Then, given a sequence of rewards received at each time step, the agent's goal is to maximize the expected return, where the return G_t is defined as a function of the reward sequence. The simplest return is given by the sum of future rewards, but it does not hold if there are infinite time steps. Hence, it's better to use the discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1},$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate and T is the final time step, which could be finite or infinite. The sequence of time steps from 0 to T is called episode.

The discount rate determines the present value of future rewards: a reward received in the future is less worth than what it would be worth if it were received immediately. Discounting observations that are far in time has more sense in non-stationary environments than in stationary ones. If $\gamma < 1$, the series converges as long as the reward sequence $\{R_k\}$ is bounded, whereas if $\gamma = 0$, the agent is "myopic" since it maximizes only the immediate rewards. While γ approaches to 1, future rewards are taken into account more strongly, until the case of $\gamma = 1$ in which all rewards are equally important, but in this case, the sum is defined if and only if $T < \infty$.

The problem described is a decision process, and it's called Markov decision process if its environment has the Markov property: the environment's response at $t + 1$ depends only on the state and action at t and not on all the states of previous time steps. Even if the state hasn't the Markov property, it's possible to approximate it as a Markov state. Moreover, if the state and action sets are finite, the problem is a finite Markov decision process, which will be the problem considered in the following.

Any particular finite Markov decision process is defined by its state and action sets and by the one-step dynamics of the environment, which are specified by the probability of each possible pair of next state s' and reward r , given any state s and action a :

$$p(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

Chapter 3. Reinforcement Learning

Given these dynamics, it's possible to compute other relevant quantities of the environment:

- the expected rewards for state-action

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

- the state-transition probabilities

$$p(s' | s, a) = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

- the expected rewards for state-action-state triples

$$r(s, a, s') = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r p(s', r | s, a)}{p(s' | s, a)}$$

Value functions and optimality

Many reinforcement learning algorithms involve estimating value functions: functions of states (or state-action pairs) that estimate how good it is for the agent to be in a given state (or to perform a given action in a given state). The value of a state s under a policy π is the expected return of being in state s and following the policy π and for Markov decision processes it can be written as

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

where \mathbb{E}_π denotes the expected value of a random variable given that the agent follows a policy π and t is any time step. The function v_π is called state-value function for policy π .

Similarly, the value of taking action a in state s under a policy π is the expected return of being in state s , taking action a and following the policy π :

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

The function $q_\pi(s, a)$ is called action-value function for policy π .

Both value functions can be estimated from experience and they satisfy recursive relationship between the value of any state s and the value of its possible successor states, for any

policy π :

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma v_\pi(s')] \quad \forall s \in \mathcal{S}
\end{aligned} \tag{3.1}$$

This equation is called Bellman equation for v_π and it states a relationship between the value of a state and the values of its successor states, where the value function v_π is the unique solution.

Since solving a reinforcement learning problem means finding a policy able to achieve the maximum of reward over the long run, these equations are useful to find an optimal policy for a finite Markov decision process. A policy π is said to be better than or equal to another policy π' if v_π is greater than or equal to $v_{\pi'}$ for all states. Then the optimal policy π_* is the one which is better than or equal to all the other policies. This policy always exists and it could be not unique, but this doesn't influence the solution, since they are equivalent in terms of state-value function and action-value function.

In fact, the optimal state-value function is

$$v_*(s) = \max_{\pi} v_\pi(s) \quad \forall s \in \mathcal{S}$$

and the optimal action-value function is

$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

This last function gives, for each pair (s, a) , the expected return for taking action a in state s and thereafter following an optimal policy. Thus, q_* can be written in terms of v_*

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

Since v_* is the value function for a policy, it satisfies the Bellman equation and for optimality conditions it can be written without reference to any specific policy. Therefore, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must

equal the expected return for the best action from that state:

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}_{\pi_*} [G_t | S_t = s] \\
&= \max_a \mathbb{E}_{\pi_*} [R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E} [R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')] \\
&= \max_{a \in \mathcal{A}(s)} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) v_*(s') \right) \tag{3.2}
\end{aligned}$$

The Bellman optimality equation for q_* can be computed as follows:

$$\begin{aligned}
q_*(s, a) &= \mathbb{E} [R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')] \\
&= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \max_{a' \in \mathcal{A}} q_*(s', a') \tag{3.3}
\end{aligned}$$

The Bellman optimality equations give the optimal solution and are used to solve analytically a finite Markov decision process (by dynamic programming methods such as value iteration or policy iteration [12]) and then to solve a reinforcement learning problem.

In general, however, the analytical solution is computationally expensive and assume the perfect knowledge of the dynamics of the environment (which is not true in real scenarios), hence there are other solution algorithms that are more efficient. These algorithms are approximate methods and try to estimate the optimal action-value function $q_*(s, a)$, which simply leads to the optimal policy: select the action that maximizes $q_*(s, a)$.

3.2 Simple case: multi-armed bandit problem

This section will explore some challenges of reinforcement learning problems in a particular setting: the simple version of the multi-armed bandit problem.

This problem considers an agent facing repeatedly with a choice among k different actions. It is actually the same choice which has to be taken again at every time step. After each choice, he will receive a reward, chosen from a stationary probability distribution that depends on the selected action. The agent's objective is to maximize the expected total reward over some time period (for example, 1000 time steps).

It's easy to observe that a general Markov Decision Problem can be seen as a sequence of multi-armed bandit problems. At each time step, the agent could choose an action from a set of k actions and, after selecting one of them, he finds a new environment's state. In this new state, the agent faces again with a multi-armed bandit problem, but with a different set of k actions depending on the new state. In this way, at the end of the period, the agent will have seen a sequence of multi-armed bandits.

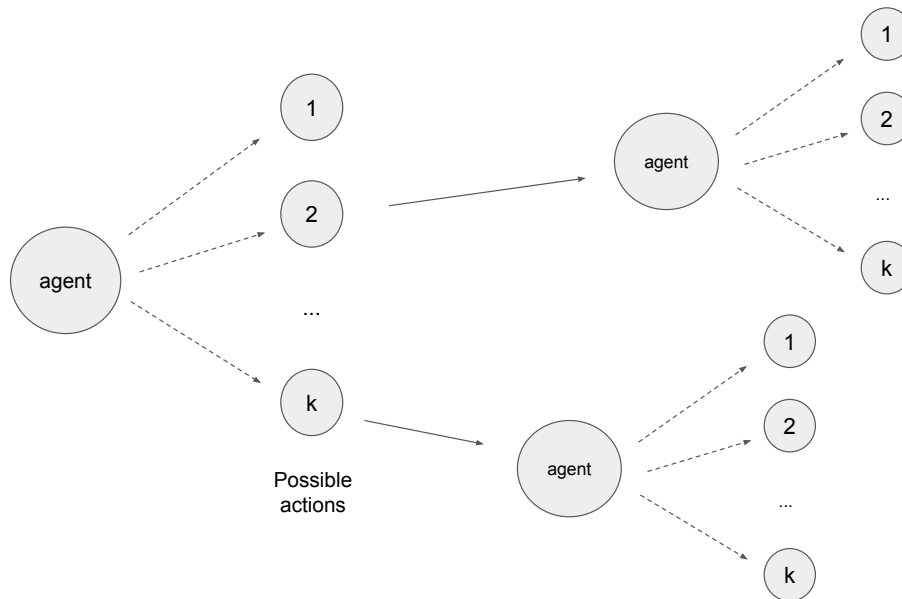


Figure 3.2: Representation of Markov Decision Process as a sequence of multi-armed bandit problem.

3.2.1 Exploration-exploitation trade-off

As shown before, an important challenge of reinforcement learning is to find a way of balancing exploration of unknown action's value and exploitation of current knowledge. In fact, reinforcement learning evaluates the actions that the algorithm tries rather than learning from instructions about the correct actions.

In the multi-armed bandit problem a value is associated to each of the k possible actions. This value represents the expected reward given that that action is selected. Denoting the action selected on time step t as A_t and its reward as R_t , then the value of an arbitrary action a is

$$q_*(a) = \mathbb{E}[R_t | A_t = a]$$

If the action value is known by the agent, then he would always select the action with the highest value and trivially solve the problem. However, since values are always uncertain, it's impossible to know the exact action value, and what the agent knows is only an estimate, denoted as $Q_t(a) \approx q_*(a)$.

Collecting action value estimates, at any step, there will be at least one action whose estimated value is greatest and this action is called *greedy action*. When the agent selects a greedy action, he is said to exploit his current knowledge, otherwise, when he selects a non-greedy action, he is said to explore the environment in order to improve the estimations. In practice, exploitation is the right thing to do to maximize the immediate expected reward, while exploration allows future exploitation moves to be chosen with a wider comprehension of the value function.

This is the well-known conflict between exploration and exploitation and its solution is different in any specific case and it depends on the precise values of the estimates, uncertainties and the number of remaining steps. Then, balancing exploration and exploitation can be done in many possible ways that are simple or more sophisticated. In the case of multi-armed bandit problem, it's very simple to understand the idea of these methods.

Since the true value of an action is defined as the mean reward when that action is selected, it's possible to estimate this value by averaging the observed rewards (i.e. using the sample average):

$$Q_t(a) = \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a}}$$

where $\mathbf{1}$ is the indicator function.

Greedy action selection methods

The simplest action selection rule is to select at each step the action (or one of the actions) with highest estimated action value. This is a *greedy action selection method* and it can be written as

$$A_t = \underset{a}{\operatorname{argmax}} Q_t(a)$$

This method always exploits current knowledge to maximize immediate reward and it never tries any non-greedy action to see if it might be better.

A simple alternative is to behave greedily most of the time, but every once in a while, with a small probability ϵ , to select randomly from all the actions, independently of the action value estimation. This rule is called *ϵ -greedy action selection method*. With this method, it's possible to exploit current knowledge most of the time and to explore sometimes some other actions, in order to improve the knowledge for the subsequent exploitation step. It can be proved that, in the long run, ϵ -greedy method performs better than the greedy one [12].

These methods based on the average of the observed reward are efficient in a stationary environment, but not if the problem is changing over time, i.e. it is non-stationary. In these cases, it makes sense to weight recent rewards more than past ones, since, if environment changes over time, past observations do not reflect actual state as well as recent observations. This could be done introducing a parameter that implies an exponential decay of weights [12].

Optimistic initial values

All of these mean-based action selection methods are in some sense dependent on the initial action-value estimates, defined by the user, in order to start the algorithm. This could turn out in a distortion of the result, especially in a non-stationary environments. However, initial action values can also be used to encourage exploration in a simple way: for example, setting them to very optimistic values. In this way, all of the actions are tried at least once, since after selecting an action, the value of that action decreases. This technique is called *optimistic initial values* and it can be shown that at the beginning it's slower than greedy methods, but in the long run, it finds a better optimal solution.

Upper confidence bound action selection

Another improvement for greedy methods is to select among the non-greedy actions the action with maximum potential for being optimal, observing uncertainties in the estimates and how the estimates are close to being maximal. This can be done selecting action as

$$A_t = \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right] \quad (3.4)$$

where $N_t(a)$ is the number of times that the action a is selected prior to time t and c is a positive parameter controlling the degree of exploration.

This method is called *upper confidence bound (UCB) action selection*. Its idea is to consider, in the maximization, a measure of uncertainty (variance) in the estimate of an action value, captured by the square-root term. Performances of the UCB method are better than the ϵ -greedy one, but it is more difficult to generalize it to a general setting, beyond the multi-armed bandit problem.

Gradient Bandit algorithm

The last example is quite different from the other presented so far since it has no interpretation in terms of reward. In fact, instead of estimating action values, this method, called *Gradient Bandit algorithm*, learns a numerical preference $H_t(a)$ for each action. The preference function is updated at each time step: if the action a is selected, the preference of action a increases of a factor proportional to the received reward, while the preference of all the other actions decreases of another factor, proportional to reward too. Using the preference function, this method selects the action with larger action probability defined according to a Boltzmann distribution:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}}$$

In practice the method selects more often the action with larger relative preference over others. This method performs very well if the best action is well-separated by the others but it is very slow to converge to the optimal solution.

Moreover, using as preference function one of the previous way of computing the Q-values or the objective function to be maximized, it's possible to use the Gradient Bandit algorithm as a stochastic way of selecting actions.

3.2.2 General solution algorithm

In the simple setting of multi-armed bandit problem, it's possible to define the general form of a solution algorithm, after studying the incremental implementation of the action-value methods described above. For each of the k possible actions, the reward received after the i^{th} selection of this action is denoted by R_i and the estimate of the action value after it has been selected $n - 1$ times is

$$Q_n = \frac{R_1 + R_2 + \dots + R_{n-1}}{n - 1}$$

In order to select the best action, at each time step, the agent has to compute Q_n for each possible action. A computationally efficient way of doing this is to consider the following

3.2. Simple case: multi-armed bandit problem

incremental updating of Q_n :

$$Q_{n+1} = \frac{1}{n} \sum_{i=1}^n R_i = Q_n + \frac{1}{n} [R_n - Q_n]$$

Therefore, the pseudocode for the bandit algorithm using the ϵ -greedy action selection is the following Algorithm 1, where $Q(a)$ is the estimated action value for action a , $N(a)$ is the number of times the agent has selected action a , $bandit(a)$ is a function that taking an action a returns a corresponding reward.

Algorithm 1 ϵ -greedy solution algorithm for multi-armed bandit problem

```
Initialize, for  $a = 1$  to  $k$ :  
   $Q(a) \leftarrow 0$   
   $N(a) \leftarrow 0$   
repeat  
   $A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1 - \epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{cases}$   
   $R \leftarrow bandit(A)$   
   $N(A) \leftarrow N(A) + 1$   
   $Q(A) \leftarrow Q(A) + \frac{1}{N(A)} [R - Q(A)]$   
until forever
```

The main step of this algorithm is the updating rule for the action value estimate, which is in the general form

$$NewEstimate \leftarrow OldEstimate + StepSize [Target - OldEstimate]$$

where *Target* is the desirable estimate and *StepSize* is the parameter that indicates the length of each time step.

Any solution algorithm for a Markov decision process can be seen as an extension of this simple algorithm, where the estimated value function is denoted by $Q(s, a) \approx q_*(s, a)$.

3.3 Examples of solution algorithms

This section will present some solution algorithms for the reinforcement learning problem in the form of a finite Markov decision process. These algorithms can be divided into two main classes:

- model-based methods: the agent learns a model of the environment and then uses it to derive the optimal policy.
- model-free methods: the agent learns the optimal policy without learning a model of the environment;

In the context of Markov decision processes, the model consists in the knowledge of the state-transition probability function $p(s'|s, a)$ and of the expected reward function $r(s, a)$. From (3.3), it's easy to see that knowing these functions leads directly to the action-value function. Hence, in order to find the optimal policy, both class of methods try to estimate the function $q_*(s, a)$, but model-based methods do it by the estimation of functions $p(s'|s, a)$ and $r(s, a)$, while model-free methods estimate q_* directly.

A wider and more exhaustive discussion about possible solution algorithms can be found in [12] and [8]. In the following, there is just a selection of these methods in order to understand the idea behind reinforcement learning methods and to explore the most inspiring algorithms for this work.

3.3.1 Exact solution methods

Before describing the approximate algorithms, it's worth to explore dynamic programming techniques for determining the optimal policy given the correct model. Given a finite Markov Decision Process, these algorithms assume the perfect knowledge of the model describing the environment, hence they are not effective in the real case studies. However, these techniques will serve as foundation and inspiration for the other algorithms.

The main idea is that the dynamic programming algorithm can be obtained by turning the Bellman optimality equations into update rules for improving approximations of the desired value functions. Indeed, given the optimality equation for the state value function (3.2), the optimal policy can be specified as

$$\pi_*(s) = \operatorname{argmax}_a \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v_*(s') \right) \quad (3.5)$$

Thus, a possible solution algorithm is the policy iteration, described in Algorithm 2. At each iteration, it consists of two main steps: the evaluation of the actual policy and the improvement of the policy.

The policy evaluation is done using the Bellman equation 3.1, where the value function of a policy is computed as the expected reward that will be gained, at each state, by executing that policy. Then, the value function $V \approx v_*$ can be estimated by solving a set of linear equations and this can be done using the following iterative form:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_k(s')]$$

After the evaluating step, the algorithm tries to improve the policy by changing the action that is expected to improve more the value function, according to 3.5.

Algorithm 2 Policy Iteration Algorithm

initialize value function $V(s) \in \mathbb{R}$ and policy $\pi(s) \in \mathcal{A}(s)$ arbitrarily $\forall s \in \mathcal{S}$

repeat

policy-stable \leftarrow *true*

$V = \text{EVALUATEPOLICY}(\pi, V)$

$\pi = \text{IMPROVEPOLICY}(\pi, V)$

until *policy-stable*

return best policy $\pi \approx \pi_*$ with value $V \approx v_*$

function EVALUATEPOLICY(π, V)

repeat

$\Delta \leftarrow 0$

for $s \in \mathcal{S}$ **do**

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end for

until Δ small enough

end function

function IMPROVEPOLICY(π, V)

for $s \in \mathcal{S}$ **do**

$old\text{-}action \leftarrow \pi(s)$

$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} (r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) V(s'))$

if $old\text{-}action \neq \pi(s)$ **then**

policy-stable \leftarrow *false*

end if

end for

end function

The policy iteration algorithm takes few iteration to converge, but each iteration takes a long time due to the policy evaluation step. However, from 3.5, it can be seen that finding the optimal policy is equivalent to find the optimal value function an then derive the optimal policy.

Chapter 3. Reinforcement Learning

This idea is captured by the value iteration algorithm, presented in Algorithm 3, that uses the following updating rule for the value function at each iteration:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$$

Algorithm 3 Value Iteration Algorithm

```
initialize  $V(s)$  arbitrarily  $\forall s \in \mathcal{S}$ 
repeat
   $\Delta \leftarrow 0$ 
  for  $s \in \mathcal{S}$  do
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma v_k(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
  end for
until  $\Delta$  is small enough
return a policy  $\pi \approx \pi_*$  such that 3.5 holds
```

This algorithm converges to the optimal policy in more iteration than the policy iteration, but each iteration takes less time. Finally, both methods find a solution to the reinforcement learning problem assuming the perfect knowledge of the environment's model.

These algorithms could be too computationally expensive and they're not worth to be improved more since they cannot be used in practice for the assumption of perfect knowledge of the environment. In fact the perfect knowledge is impossible to be achieved in real applications. However, the ideas of these methods are used as foundation for the approximate methods.

3.3.2 Model-based algorithms

As explained above, in model-based algorithms the primary focus of the learning experience is to build a model of the environment by estimating the state-transition probabilities and the expected rewards.

DYNA algorithm

In the Dyna algorithm [8], the agent simultaneously uses experience to build a model and to adjust the policy, and uses the model to adjust the policy. In practice, at each interaction with the environment, given an experience tuple (s, a, s', r) the following steps are performed:

- Update the model by incrementing statistics for observing transition from s to s' on action a and for receiving reward r for taking action a in state s . Denote the updated models by \hat{p} and \hat{r} .

- Update the policy at state s based on the updated model using the rule

$$Q(s, a) = \hat{r}(s, a) + \gamma \sum_{s'} \hat{p}(s'|s, a) \max_{a'} Q(s', a')$$

- Choose k state-action pairs at random. Update them again according to the same rule as in the previous step.
- Choose an action a' to perform in state s' , based on the Q values but eventually modified by an exploration strategy.

Even if the convergence of this algorithm is very fast, sometimes, rather than concentrating on the "interesting" part of the state space, it wastes time in updating random state-action pairs, especially when it has already reached the goal or when the agent is stuck in a dead end.

Prioritized sweeping

A possible solution to the problems above is given by the prioritized sweeping, a modified version of Dyna in which updates are not chosen at random and which uses $V(s) \approx v_*(s)$ as value function, instead of $Q(s, a)$ (that is values are associated only with states and not with state-action pairs). Moreover each state remembers its predecessors (states with non-zero transition probability to it under some action) and has a priority, initially set to zero.

Hence, instead of updating k random state-action pairs, prioritized sweeping updates k states with the highest priority as follows:

- Store the current value of the state: $V_{old} = V(s)$
- Update the state's value

$$V(s) = \max_a (\hat{r}(s, a) + \gamma \sum_{s'} \hat{p}(s'|s, a) V(s'))$$

- Set the priority of the state to 0, compute the value change $\Delta = |V_{old} - V(s)|$ and use it to modify the priorities of the predecessors of s .

This last step is used to spread information of one state to its predecessors and its useful to direct the algorithm in an interesting part of the space (if value change is large) or to avoid exploring states that bring no further improvement to the state value. As expected, this algorithm is faster than Dyna to reach the convergence to the optimal policy.

3.3.3 Model-free algorithms

A large class of solution algorithms is known as temporal difference methods. The idea comes from the fact that, in general, an action could not have an immediate reward, but only a future reward, depending also on subsequent actions. Hence, the problem is to find a way for determining whether the action just taken is good or could have other effects.

One strategy for doing this is to wait until the "end" and reward the actions taken if the result was good and punish them if the result was bad. However, since it's not easy to know what the "end" is, this class of solutions extends the exact iterative algorithms in order to improve the efficiency.

The first strategy is the adaptive heuristic critic algorithm, which is a version of policy iteration in which the value-function computation is not implemented by solving a set of linear equations, but it is computed by an algorithm called TD(0). This approach consists of two component: a critic (AHC) component and a reinforcement learning (RL) component, as represented in figure 3.3.

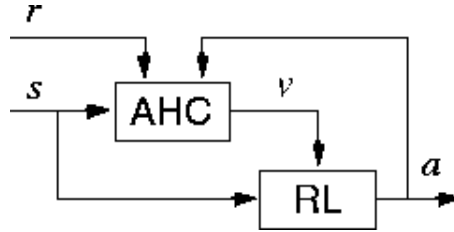


Figure 3.3: Architecture for the adaptive heuristic critic algorithm.

The reinforcement-learning component is a simple multi-armed bandit algorithm, modified to deal with multiple states and non-stationary rewards. However, instead of maximizing the instantaneous reward, it maximizes the heuristic value v , computed by the critic. The critic uses the real external reinforcement signal to map states to their expected discounted values given that the policy being executed is the one currently instantiated in the RL component.

The analogy with policy iteration algorithm can be seen if the components are imagined as working in alternation: the policy π implemented by RL is fixed and the critic learns the value function V_π for that policy. Then, fixing the critic, the RL component learns a new policy π' that maximizes the new value function, and so on.

Therefore, the computation of the value of the policy is performed by the critic: given an experience tuple (s, a, r, s') that summarizes a single transition in the environment, the TD(0) algorithm uses the update rule

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

In practice, whenever a state s is visited, its estimated value is updated to be closer to $r + \gamma V(s')$, since r is the instantaneous reward received and $V(s')$ is the estimated value of the actual next state.

The TD(0) rule is actually an instance of a more general class of algorithms called TD(λ), with $\lambda = 0$. The general TD(λ) is

$$V(u) = V(u) + \alpha(r + \gamma V(s') - V(s))\epsilon(u)$$

and it's applied to every state according to its eligibility trace $\epsilon(u)$, rather than just to the immediately previous state s . The eligibility trace of a state s is the degree to which it has been visited in the recent past and it is defined as

$$\epsilon(s) = \sum_{k=1}^t (\lambda\gamma)^{t-k} \delta_{s,s_k} \quad \text{where } \delta_{s,s_k} = \begin{cases} 1 & \text{if } s = s_k \\ 0 & \text{otherwise} \end{cases}$$

Hence, when a reinforcement is received, it is used to update all the states that have been recently visited, according to their eligibility. Moreover, the eligibility trace can be updated online as

$$\epsilon(s) = \begin{cases} \gamma\lambda\epsilon(s) + 1 & \text{if } s = \text{current state} \\ \gamma\lambda\epsilon(s) & \text{otherwise} \end{cases}$$

In the following, two methods of the temporal difference class will be described. Both methods have a version with eligibility trace, where the updates are done only on a small set of states, according to the eligibility trace.

SARSA

The Sarsa algorithm is a temporal difference method, trying to learn, using the TD method, the action-value function instead of the state-value function. In particular, it estimates the function $Q(s, a) \approx q_\pi(s, a)$ for the current policy π and for all states s and actions a , using the TD(0) method, described above.

The methods observes transitions from state-action pair to state-action pair and learn values of pairs. For each experience tuple (S, A, R, S', A') , the updating rule becomes:

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$$

This rule is applied to every transition step of any episode observed, until the terminal state is reached, where the Q-value is zero. Hence, the algorithm for the general Sarsa method is given in Algorithm 4.

Chapter 3. Reinforcement Learning

The convergence of the Sarsa algorithm depends on the nature of the policy dependence on Q . Indeed, the Sarsa method is an on-policy method, which means that policy is improved while decisions are made, hence it changes every time Q changes.

Algorithm 4 Sarsa Algorithm

```
initialize arbitrarily  $Q(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$  and  $Q(\text{terminal-state}, \cdot) = 0$ 
repeat
  initialize  $S$ 
  choose  $A$  from  $S$  using policy derived from  $Q$ 
  repeat
    take action  $A$  and observe  $R, S'$ 
    choose  $A'$  from  $S'$  using policy derived from  $Q$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'$ 
     $A \leftarrow A'$ 
  until  $S$  is a terminal state
until all episodes are observed
```

Q-LEARNING

The idea of the Q-learning algorithm is similar to the Sarsa algorithm, but Q-learning is an off-policy method, so that the learned action-value function Q directly approximates the optimal q_* , independent on the policy being followed. However, the policy has still an effect in determining which state-action pairs are visited and updated: it selects the state-action pair with maximum Q value for the current state.

Hence, given an experience tuple (S, A, R, S') , the Q-learning rule is

$$Q(S, A) = Q(S, A) + \alpha(R + \gamma \max_a Q(S', a) - Q(S, A))$$

The algorithm of Q-learning is presented in Algorithm 5, where convergence of Q values to q_* is guaranteed if each action is executed in each state an infinite number of times on a infinite run and α is decayed appropriately.

Finally, the optimal policy is derived from the learned function $Q \approx q_*$, since $\pi_*(s) = \arg\max_a q_*(s, a)$. As in all the off-policy methods, the optimal policy is different from the one used for collecting data, which could be any of the exploration-exploitation methods presented for the multi-armed bandit problem. However, Q-learning is exploration insensitive: Q values will converge to the optimal values, independent on how the agent behaves while collecting data. Hence the exploration method used does not affect the convergence.

Algorithm 5 Q-learning Algorithm

```

initialize arbitrarily  $Q(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$  and  $Q(\text{terminal-state}, \cdot) = 0$ 
repeat
  initialize  $S$ 
  repeat
    choose  $A$  from  $S$  using policy derived from  $Q$ 
    take action  $A$  and observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is a terminal state
until all episodes are observed

```

3.4 Bayesian Q-learning

Bayesian methods formulate the reinforcement learning problem as a belief-state Markov Decision Process, in which an agent's beliefs are explicitly modeled as part of the state [14]. Hence, an agent can infer the exploratory value of an action, by reasoning about how the information obtained by making an action may influence better future decisions.

As seen, in the Q-learning method, the agent is uncertain about its estimate of the Q-value of each state. In the Bayesian approach of Q-learning, this uncertainty is represented using probability distributions and the agent selects actions on the basis of local Q-value information [2]. These distributions can be propagated over Q-values in order to make more informed decisions and this results in a global exploration.

The algorithm of Bayesian Q-learning is similar to the standard Q-learning algorithm, except that it stores the hyperparameters, instead of storing the Q-values.

This method will be described in details in this section, since it is the foundation for the approach proposed in this work for dynamic pricing.

3.4.1 Q-value distributions

Since the framework is Bayesian, it's necessary to consider prior distributions over Q-values and then update them based on the experience of the agent. Denoting by $R_{s,a}$ the random variable that considers the total discounted reward received when action a is performed in state s and an optimal policy is followed thereafter: then this random variable is such that $q_*(s, a) = \mathbb{E}[R_{s,a}]$.

Generally, the initial distribution of $R_{s,a}$ is unknown and it's reasonable to assume that it has a normal distribution. This assumption implies that to model uncertainty about this distribution it's sufficient to model distributions of mean $\mu_{s,a}$ and precision $\tau_{s,a} = 1/\sigma_{s,a}^2$ of the normal $R_{s,a}$. Hence, the Q-values are modeled by the mean $\mu_{s,a}$.

Another assumption that is fairly reasonable is that the prior beliefs about $R_{s,a}$ are independent of those about $R_{s',a'}$: prior distribution over $\mu_{s,a}$ and $\tau_{s,a}$ is independent of the prior distribution over $\mu_{s',a'}$ and $\tau_{s',a'}$ for $s \neq s'$ and $a \neq a'$. This assumption has not bad consequences since it restricts only the prior form of the knowledge, but it does not necessarily imply that the posterior distribution has the same independence condition.

Moreover, it's possible to assume that the prior distributions $p(\mu_{s,a}, \tau_{s,a})$ over the parameters of each $R_{s,a}$ are normal-gamma distributions. A normal-gamma distribution over the mean μ and the precision τ of an unknown normally distributed variable R is fully determined by a tuple of hyperparameters $(\mu_0, \lambda, \alpha, \beta)$. Hence, $p(\mu, \tau) \sim NG(\mu_0, \lambda, \alpha, \beta)$ if

$$p(\mu, \tau) \propto \tau^{\frac{1}{2}} e^{-\frac{1}{2}\lambda\tau(\mu-\mu_0)^2} \tau^{\alpha-1} e^{-\beta\tau}$$

Given a normal gamma distribution and given any sequence of observations (random samples of R) r_1, \dots, r_n with $M_1 = \frac{1}{n} \sum_i r_i$ and $M_2 = \frac{1}{n} \sum_i r_i^2$, the posterior distribution $p(\mu, \tau | r_1, \dots, r_n)$ is also a normal-gamma distribution with parameters

$$\begin{aligned} \mu'_0 &= \frac{\lambda\mu_0 + nM_1}{\lambda + n} & \lambda' &= \lambda + n \\ \alpha' &= \alpha + \frac{1}{2}n & \beta' &= \beta + \frac{1}{2} \left(n(M_2 - M_1^2) + \frac{n\lambda(M_1 - \mu_0)^2}{\lambda + n} \right) \end{aligned}$$

Therefore, by these assumptions, to represent the agent's prior over the distribution of $R_{s,a}$ is sufficient to keep track of the tuple $(\mu_0^{s,a}, \lambda^{s,a}, \alpha^{s,a}, \beta^{s,a})$.

This representation by storing hyperparameters can be used also for the posterior distribution, but assuming, in addition, that the agent's posteriors of two subsequent steps are independent. This assumption is not really true in Markov Decision Processes since state-action values of two steps are related by the Bellman equations, but it can be assumed anyway.

3.4.2 Action selection

Given a probability distribution over $Q(s, a) = \mu_{s,a}$ for all states s and actions a , the algorithm has to select an action to perform in the actual state, following a certain policy. A possible approach is the usual greedy-approach, that selects the action a maximizing the expected value $E[\mu_{s,a}]$, which is just the estimate of the mean of $R_{s,a}$. Hence, this method selects the action with the greatest mean, but it does not take into account any uncertainty about the Q-value.

Another action selection approach is the Q-value sampling, in which the idea is to select actions stochastically, based on the current subjective belief that they are optimal. Hence,

given the independence of posteriors, action a is performed with probability

$$\begin{aligned}\mathbb{P}(a = \operatorname{argmax}_{a'} \mu_{s,a'}) &= \mathbb{P}(\forall a' \neq a, \mu_{s,a} > \mu_{s,a'}) \\ &= \int_{-\infty}^{\infty} \mathbb{P}(\mu_{s,a} = q_a) \prod_{a' \neq a} \mathbb{P}(\mu_{s,a'} < q_a) dq_a\end{aligned}$$

Since the marginal density of $\mu_{r,s}$ is given by the density of a t-distribution², the computation of this probability can be avoided by sampling a value from each $p(\mu_{s,a})$ and then executing the action with the highest sampled value.

Finally, this method is a stochastic exploration policy in which the probability of performing an action is related to the distribution of the associated Q-value and not to the amount by which choosing an action might improve the current policy (i.e. by choosing an exploration action).

This drawback of Q-value sampling can be solved by using the Myopic-VPI selection: it is based on value of perfect information ($VPI(s, a)$), which is a quantitative estimate of the expected gains from exploration. Thus, the idea is to balance VPI against the expected cost of doing a potentially suboptimal action. Then, the agent should choose the action that maximizes $\mathbb{E}[Q(s, a)] + VPI(s, a)$, hence the exploration is encouraged by VPI when the agent is uncertain about the estimate of Q-values.

3.4.3 Updating Q-values

The last step is finding a way for updating the estimate of the distribution over Q-values after executing a transition. This step is quite complicated, since the distribution of Q-values considers expected total rewards, while the observations are about actual and local rewards.

Given an experience tuple (s, a, r, s') , it would be useful to know the complete sequence of rewards received from s' onwards. Denoting by $R_{s'}$ the random variable for the discounted sum of rewards from s' and assuming the agent follows the apparently optimal policy, it follows that $R_{s'}$ is distributed as $R_{s',a_{s'}}$ where $a_{s'}$ is the action with highest expected value at s' .

In order to use this distribution for the unknown future experiences, a possible method is moment updating: given random sample values $R_{s'}^1, \dots, R_{s'}^n$ from the distribution $R_{s',a_{s'}}$, update the posterior distribution $P(R_{s,a})$ with the sample $r + \gamma R_{s'}^1, \dots, r + \gamma R_{s'}^n$ where each sample has weight $\frac{1}{n}$.

²see [3] for details

Chapter 3. Reinforcement Learning

As seen above, to update the normal-gamma distribution, the only quantities needed are the first two moments of this sample that can be computed in the following way, as n goes to infinity:

$$M_1 = \mathbb{E}[r + \gamma R_{s'}] = r + \gamma \mathbb{E}[R_{s'}]$$

$$M_2 = \mathbb{E}[(r + \gamma R_{s'})^2] = \mathbb{E}[r^2 + 2\gamma r R_{s'} + \gamma^2 R_{s'}^2] = r^2 + 2\gamma r \mathbb{E}[R_{s'}] + \gamma^2 \mathbb{E}[R_{s'}^2]$$

Since $R_{s'}$ is distributed as a normal-gamma over mean and variance, the first two moments are computed by

$$\mathbb{E}[R_{s'}] = \mu_0 \quad \mathbb{E}[(R_{s'})^2] = \frac{\lambda + 1}{\lambda} \cdot \frac{\beta}{\alpha + 1} + \mu_0^2$$

In this way, the hyperparameters can be updated using the sample collection, where M_1 represents the expected value of the Q-values, while M_2 represents the uncertainty about this estimate.

Since usually the system could change over time and this change it's not directly included in the global expected value, it's helpful to use the exponential forgetting, in order to reduce the impact of previously seen observations on the priors by a constant at each update.

4 Adapting reinforcement learning to dynamic pricing: Liquidprice

This chapter will describe how a dynamic pricing problem can be formulated in terms of Markov Decision Process. After the theoretical framework, there will be presented the pricing algorithm implemented in the software Liquidprice. It represents a possible reinforcement learning algorithm adapted to a dynamic pricing context. Finally, this algorithm will be applied in the context of pricing flights' insurance.

4.1 General approach

As described in [10], this work will consider the following formulation of a dynamic pricing problem, concerning a perishable service. Indeed, pricing is a real-time decision-making problem in a stochastic environment and it can be modeled as a Markov Decision Process.

As seen in chapter 2, a dynamic pricing problem aims to find the optimal price in order to maximize the revenue for selling a given inventory of a product by a fixed deadline. The resulting model considers time as discrete since it's difficult to consider continuous price changes. Hence price changes are only allowed at specific time points. Assuming to have a fixed number of identical products, the price is determined by the remaining capacity and the time left before the deadline expires.

Therefore, the Markov Decision Process has the following key components:

- Time horizon $t \in T = 0, 1, \dots, m$: set of finite discrete times at which price changes occur and m is the last selling period before expiration.
- State $x_t \in \mathcal{X} = 0, 1, \dots, n$: state of the system at time t , represented by the remaining capacity.
- Action $a_t \in \mathcal{A}(x_t)$: price for the capacity at time t , chosen from the set of possible prices that can be applied when the remaining capacity is x_t .
- State transition probabilities $p_t(x_{t+1}|x_t, a_t)$: probability of having remaining capacity

x_{t+1} at time $t + 1$, given that at time t there are capacity x_t and price a_t

- Reward $r(x_t, a_t)$: expected immediate revenue gained for executing price a_t when the capacity is x_t .
- Policy π : function that specifies the price that should be set given the remaining capacity and time.
- Value function is the total expected reward, when starting at state x_t and following policy π such that $a_t = \pi(x_t)$

$$v_t^\pi(s) = \mathbb{E}_\pi \left[\sum_k r(x_k, a_k) \middle| x_t = s \right]$$

Using these elements, the reinforcement learning process is the following: at each time step, the firm, which represents the agent, observes the actual state of the market inventory and the information received about market response to different prices (reward function) and chooses the price for its product. At the subsequent time step, the firm waits for customers, observes the immediate revenue gained (i.e. how many purchases) and updates its knowledge about the market. Therefore, the problem can be solved by using any method presented in the previous section. In general, model-free methods are more effective than others in this context since it's difficult to estimate directly the dynamics of price-demand interactions.

For example, in the Q-learning method, the state-action value function $q_t^\pi(x_t, a_t)$ is given by the expected discounted total reward when starting at state x_t , choosing price a_t and following policy π . As seen before, at each iteration of the algorithm, it's necessary to store and update the Q-values for each state-action pair (x_t, a_t) . This storage operation can be done by using a tabular method (i.e. store Q-values for each pair in a matrix) or a Bayesian method as the one described for the Bayesian Q-learning (i.e. store the hyperparameters of the distribution describing the Q-values and update them using Bayesian rules).

In many applications, the use of distributions for Q-values is more effective, since it's possible to keep track not only of the estimate but also of the precision of that estimate. Hence, the selection of the best action is able to take into account the precision (or confidence) of the Q-value estimate. The need of using precision is also motivated by the fact that the knowledge of the environment is highly uncertain and it changes over time, possibly with high frequency.

4.1.1 Empirical contextual problems

The model described by the above Markov decision Process can be extended to more than one product, by simply considering the state variable as a state vector, containing the remaining capacity for each product. All the other quantities can be easily extended to their vectorial form. However, this representation models different products as if the demand of one product is independent of the demand of the others, without considering possible relations between different products. In fact, in many situations, there are products with

similar demand or with a demand that influences demand of other products since customers have to make a choice between them. This problem is usually addressed as object choice problem and it needs to be taken into account in a more complex model than the simplest Markov decision process.

As seen, the environment's response to the price set is given by the immediate revenue, which means that the customer buys or not the product at the specified price. However, this reward characterization could be insufficient to understand really what is the best price to propose, since the zero-reward might have different interpretations: it could represent a customer who is not interested in that product whatever is the price or a customer that would buy the product but the price is too high. This problem can be addressed in different ways, depending on the context.

For example in an e-commerce setting it's possible to have an idea of customers' interests by observing their searching history (if available) and then differentiate the reward: for visualized but not purchased products the reward will penalize the price proposed, while for not visualized products the reward will penalize the product itself, giving an idea of demand distributions between products. On the other hand, this approach is not effective in brick-and-mortar context since it's not easy to have direct information about the demand distributions and customers price-sensitivity.

Moreover, given the uncertainty of the market, the evaluation of the value function in order to find the optimal policy could be quite challenging. Indeed if the purchases are very rare during the selling horizon, there is not enough available data to evaluate the goodness of a certain price and then it could take very long time to have a good result. This could happen also in cases in which the horizon is infinite (non-stop selling) where obviously it's not possible to wait until the end of the horizon. Furthermore, even if reinforcement learning is adaptive to very fast changing in the environment, having few data or waiting for the end of the horizon could lead to price decisions that are related to a wrong market model.

Finally, in dynamic pricing context every decision and action taken is related to prices and people, hence the typical exploration-exploitation dichotomy could have a trade-off difficult to find. In fact, exploration actions to strange prices could lead to uncontrolled customers' reactions. On the other hand, exploitation actions that always select low prices could have very bad consequences on the business of the firm. Therefore, the dynamic pricing model has also to consider some business constraints, that are specific of any context, since not all the actions are always available.

The reinforcement learning approach for solving dynamic pricing problems is then possible if it is able to address these challenges.

The scope of this work is to present the software Liquidprice, which is actually the result of adapting reinforcement learning to dynamic pricing problems, trying to solve these challenges.

4.2 Liquidprice: a software for dynamic pricing

Liquidprice is a software developed to be a platform for dynamic pricing, available to any company for pricing its goods. In practice, every product that is going to be sold can be priced following a specific goal-directed strategy. The strategy used is based on a reinforcement learning algorithm, adapted to the specific dynamic pricing context in which it works.

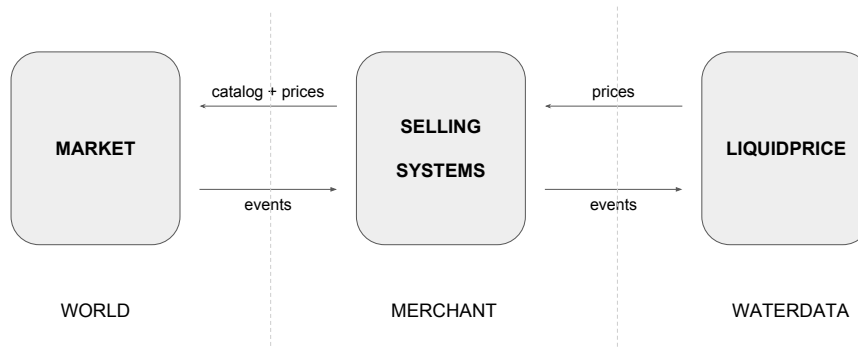


Figure 4.1: Integration of Liquidprice with merchant's selling systems.

Liquidprice is built to be integrated directly with the selling systems of the company that sells products to customers, as depicted in Fig. 4.1. In fact, the selling systems of the merchant send products' catalog and its prices to the market, where customers could buy or not those products. The events that happen in the market are then sent to the merchant, in order to store purchases data. Liquidprice is connected to the merchant in order to communicate which prices should be published in the market and it receives data that the merchant obtains from the market.

The software, in fact, is event-based: it listens to events that happen in the market, such as purchases. Every time a new event is received, it is processed and a new price is proposed. However, due to possible technological limitations in merchants' selling systems, it's not possible in every context to have events in real-time, in order to be processed by the software.

Moreover, Liquidprice can be adapted in a batch mode: it receives periodically the set of events happened in the previous period and then it simulates past events as if they were received when they happen in the market. This simulation can be done by keeping track of the time of the last event received and if the new event is older, it is processed as if it happened in the past. After processing all the events received in a single batch, the software sends the new prices based on the events observed.

Liquidprice is composed of many modules, each one with a specific purpose. This section will describe the modules related to the pricing algorithm and their application in the particular context of pricing flight insurances.

4.2.1 Main elements and terminology

In terms of classic reinforcement learning setting, the software plays the role of the agent, willing to apply the best price for each product it has, while the environment is represented by the market where products are sold. Interactions between agent and environment are managed as events: the agent generates the action event and the environment sends to the agent a market event.

In reinforcement learning terminology, the action event is actually the action selected by the agent. In dynamic pricing contexts, the action is almost always a price action so that it gives the price to set at each time instant or whenever required. On the other hand, the market event is the reward signal sent by the environment, giving an indication of the goodness of the action selected. Depending on the context and on the available data, the market event could be a purchase, with information about the revenue obtained, or just a visualization of the product, with a zero revenue.

Finally, the product or good to be priced is called item. It could be a single product with an inventory quantity or a group of very similar products that can be priced in the same way. Each item has all the information about its state, such as the remaining inventory quantity, or other features that characterize the specific item.

The functionality of Liquidprice, as depicted in figure 4.2, is the result of the work of three main components: a set of market response models, an information propagation engine used to build such models and a pricing policy which decides an optimal price based on models. The next sections will present how each component works and how it is applied in a real use case.

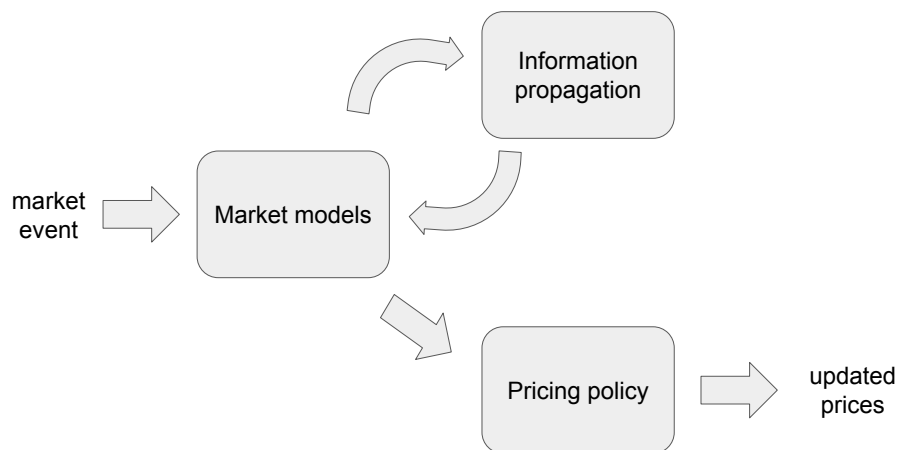


Figure 4.2: Modules interactions for Liquidprice functionality.

4.2.2 Application context

In order to see how the Liquidprice algorithm works in a real use case, we will consider the problem of pricing insurances for flight tickets. The scenario is based on a real use case in which Liquidprice is used by a company that sells flight tickets through a website platform. During the purchase process, in addition to the flight ticket, the customers can choose ancillary services such as insurances, web check-in services, airport parking or rental cars. Here, we will only focus on the pricing of insurances, which the company wants to improve in order to optimize margins.

The item for which we should propose an insurance price is represented by flights with same origin and destination and same flight price level. Clearly, the number of insurances that can be sold on a particular flight is related to the number of possible seats, but if we consider all the flights belonging to the same item, we can see that the number of insurances that are on sale at the same moment is very large and it can be considered unlimited. Hence, each item hasn't an inventory level or any other state variable, since the product that it sold is always the same and purchases are independent on other purchases.

The insurance price of an item is given by a percentage of the total flight price (called *flat value*) and for business reasons, this percentage should be in the range of $[5, 30]$. However, for completeness of the study, we will consider $[0, 30]$ as range of possible flat values.

The market events received from the environment are given by flight bookings that could be with or without an insurance purchase. If there is the insurance purchase, the reward will be positive, otherwise, the reward will penalize the information received with that event.

Given these elements and by the fact that the objective function has to maximize an immediate reward, the problem of pricing insurances can be seen as a multi-armed bandit problem, in which the agent selects the action that maximizes the immediate reward (and not a delayed reward, as in Markov Decision Process).

4.3 Market response models

For each item, Liquidprice stores the information received through the market event in a market response model. It should not be interpreted as a model of the environment in the sense of reinforcement learning terminology since the solution algorithm is model-free. Actually, the solution algorithm is based on Bayesian Q-learning and the market response model is just the representation of the state-action value function for each item (Q-value).

As in Bayesian Q-learning, the Q-values are stored as distributions, which we will refer to as value distributions, in order to have information about the estimate and the precision. Depending on the context, the value distributions could be of different types (normal-gamma, beta, ...) but in any case, it's sufficient to store and update the hyperparameters in order to have the knowledge of all that particular distribution. Using a Bayesian approach, each distribution can be updated using the market events as observations of that particular distribution. Hence, the hyperparameters are updated with the Bayesian rules.

In this way, a single event influences only one value distribution, the one related to the state-action pair of that event. However, since actions are in fact prices, given the same state, the value distribution relative to one price is related to those relative to close prices. This relation could arise also for state variables. Then, each event could possibly have effects on more than one distribution. Moreover, it could be necessary to predict Q-values for state-action pairs close to others but that have never been observed.

A way to overcome these issues is the idea of Gaussian Process Regression¹: the variables defining the state-action pairs are associated to a joint kernel function that helps to have a continuous function interpolating the real observations and giving a confidence interval for other points' estimates. For efficiency reasons the space of state-action pairs is discrete (instead of a continuous set), then in Liquidprice, it is not possible to apply the real Gaussian Process Regression, but just a local discrete interpolation and approximation (instead of a global interpolation).

Assuming independence among dimensions, the joint kernel function is given by the product of the marginal kernel functions associated with each variable defining the state-action pair. The marginal kernel function has to describe how a value is related to close values. Then the information received by an event is used not only to update the Q-value distribution of the state-action pair associated with that event but also to update all the other distributions with an attenuated information.

The attenuation factor is given by the evaluation of the joint kernel function in those state-action pairs. Then, the joint kernel function has value in $[0, 1]$: as the value of the kernel goes to 0, the distribution associated is less influenced by that event, while, as the kernel value goes to 1, there is the maximum influence.

¹all details in [11] or [9]

Finally, the market response models should consider the market changes during time (i.e. the environment is non-stationary). For this purpose, the model keeps also track of the time at which it has observed the last event (this is the time at which the event takes place, not the time at which it has been observed by the system). Therefore, at every update of the distribution, past observations are scaled by an exponential decay coefficient that assigns less importance to old observations in order to have a model for the actual environment and not an old one.

If t is the time at which the last observation was made and t_e is the time of the current event, the temporal decay coefficient is computed as

$$d = e^{-\alpha|t-t_e|}$$

where α determines how important events in the past should be.

The value of α could be computed as a function of the time interval at which the system wishes to have past observations halved (i.e. $d = 0.5$). Hence, denoting this time interval as h (halving time), the coefficient α is

$$\alpha = \frac{\log(2)}{|h|}$$

For different values of halving time, the behavior of coefficient d is depicted in Fig. 4.3.

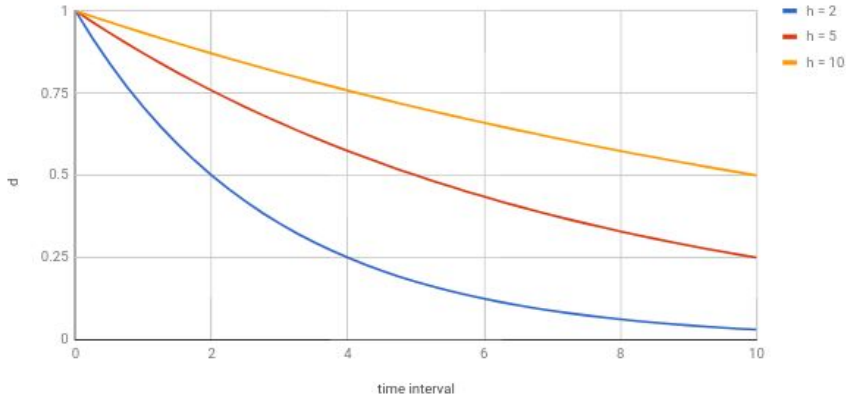


Figure 4.3: Temporal decay coefficient for different values of the halving time parameter h .

4.3.1 Kernel functions

The marginal kernel function (i.e. the kernel function associated to each single dimension) could be of many different shape depending on the specific dimension. Denoting with x the independent variable that represents the dimension of the state-action pair and with m the value of this dimension in the current event, the kernel function is in general of the form $f(x, m; k_1, \dots, k_n)$, where k_i are specific parameters of each kernel.

Gaussian kernel function

The most common kernel function is a modified version of the density of a normal probability distribution: it represents the fact that a value has influence only on a specified neighborhood around that value. The usual form of this density function is

$$\phi(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (4.1)$$

where μ is the mean and σ is the standard deviation of the distribution.

In order to use the normal distribution as a kernel, it should be modified in such a way that $f(\mu; \mu, \sigma) = 1$ and it should hold $\mu = m$ since the kernel must have its maximum value in m . Moreover, the parameter σ represents how large the influence of value x is on its neighbors. Hence, the *Gaussian kernel* is defined as

$$G(x, m; \sigma) = e^{-\frac{(x-m)^2}{2\sigma^2}}$$

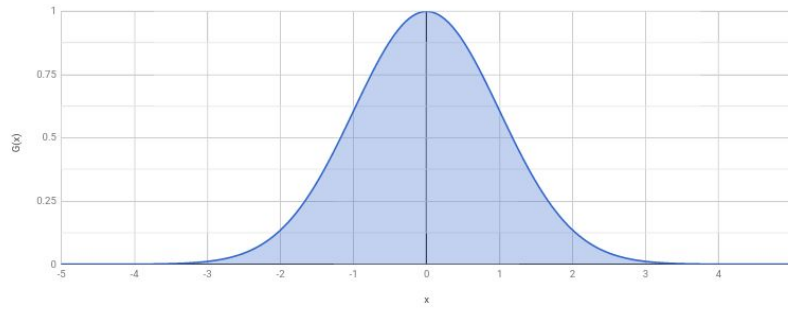


Figure 4.4: Gaussian kernel function with $m = 0$ and $\sigma = 1$.

Usually, the Gaussian kernel is associated to state dimensions. However, the dimension that has the most effect of influence is the price dimension. Depending on the perception of price influence, it is possible to define kernel functions of different shapes.

Sigmoid kernel function

First of all, it's possible to simulate the fact that if a customer has bought an item at a certain price m , he could also have bought it at all lower prices, but not at higher prices. This can be done using the indicator function:

$$I(x, m) = \begin{cases} 1 & \text{if } x \leq m \\ 0 & \text{if } x > m \end{cases}$$

However, this function is not continuous and does not take into account the uncertainty:

in fact, with the indicator function, we are assuming that if a customer has bought something at a price of 10.2, he surely would not have bought it at 10.5 or he surely would have bought at 10. Of course, this cannot be true in real markets, hence we should modify the indicator function in order to smooth the jump corresponding to m .

For this purpose, we use the *sigmoid function*, whose equation is

$$S(x, m; a, b) = \frac{1}{1 + e^{-a(x-b-m)}}$$

where b is the distance from m at which the function has value $\frac{1}{2}$ (i.e. the function is halved) and a is a parameter related to how steep the change from the minimum to the maximum value is.

The coefficient a is computed such that

$$S(m, m; a, b) = c \quad \Longleftrightarrow \quad a = \frac{1}{b} \log\left(\frac{1}{c} - 1\right)$$

From these relations, it follows that $c \in (0, 1)$ and the following situations arise:

- if $\{b > 0, c \in (0, \frac{1}{2})\}$ or $\{b < 0, c \in (\frac{1}{2}, 1)\}$, the sigmoid function is increasing and it represents the fact that customers could not have bought at higher prices, but at lower prices;
- if $\{b > 0, c \in (\frac{1}{2}, 1)\}$ or $\{b < 0, c \in (0, \frac{1}{2})\}$, the sigmoid function is decreasing and it represents the opposite situation in which if customers did not buy at a certain price, they would not have bought at higher prices.

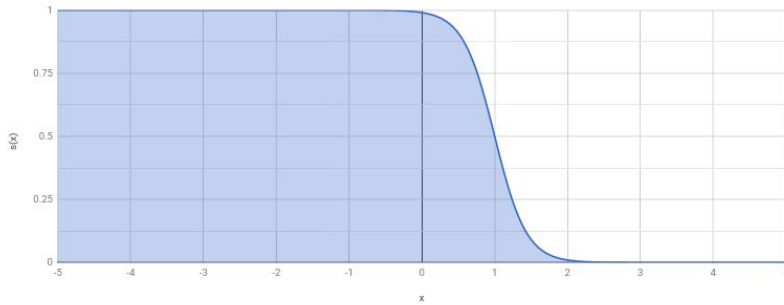


Figure 4.5: Sigmoid kernel function with $m = 0$, $b = 1$ and $c = 0.99$.

Moreover, in some contexts where prices are selected from a limited range, it's possible to assume that if a customer has bought at a certain price, he might have bought with a small probability at any higher price in that range (some customers would buy a good at any price if they really need it). The opposite assumption is valid for customers who didn't buy any good: with a certain probability, they wouldn't have bought it at any lower price in the bounded range (if they don't want a product, they wouldn't buy it at any price).

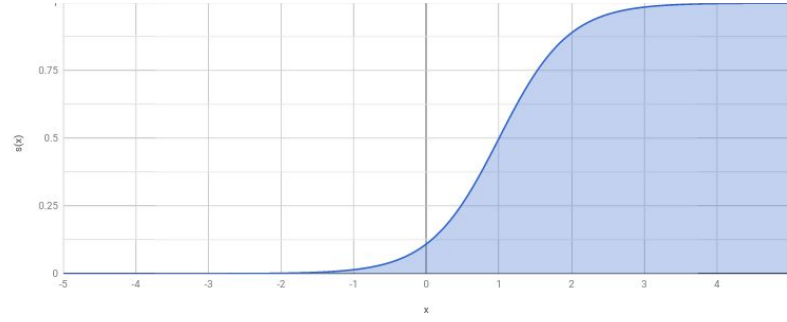


Figure 4.6: Sigmoid kernel function with $m = 0$, $b = 1$ and $c = 0.11$.

This assumption can be taken into account by composing the sigmoid kernel function described above with the *Higher Floor function*

$$HF(x; f) = x(1 - f) + f$$

where f is the small probability of buying at any price or of not buying at any price.

Combining the Higher Floor function with the sigmoid function, we obtain the Higher Floor Sigmoid kernel function

$$HFS(x, m; a, b, f) = \frac{1}{1 + e^{-a(x-b-m)}}(1 - f) + f$$

In order to find reasonable values for the parameter f , it's possible to consider the fraction of customers that buy a product at very high prices (for the decreasing kernel) and the fraction of customers that do not buy at very low prices (for the increasing kernel). These estimates can be computed by having always in the catalog products at high prices and products at low prices (possibly different products at different times). This strategy is commonly used by companies in order to evaluate, in bounded ranges of prices, probabilities of buying anyway or not buying anyway.

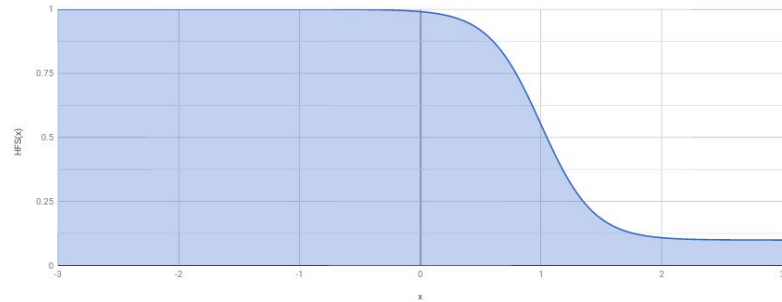


Figure 4.7: Sigmoid Higher Floor kernel function with $m = 0$, $b = 1$, $c = 0.99$ and $f = 0.1$.

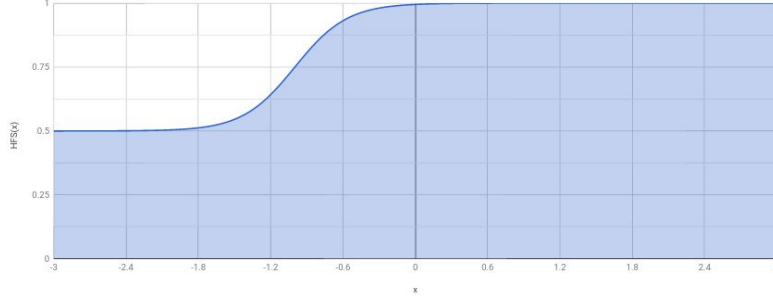


Figure 4.8: Sigmoid Higher Floor kernel function with $m = 0$, $b = -1$, $c = 0.99$ and $f = 0.5$.

Left Hat kernel function

In contexts in which the price plays a role of quality perception, it's not possible to use the sigmoid kernel function. In these cases, if the price of an item is too low, its quality is quite bad and then that item is not desirable by customers. This situation can be described by a function called *left hat* which is the composition of two normal functions, linked by a straight line.

The influence of price p decreases with decreasing price, but only until a certain value, which is the minimum price at which a customer could have bought. Moreover, the slope of the straight line gives a sort of probability that the customer would have bought at that price. The slope is decreasing as price decreases since there is uncertainty about probability for not observed prices performances.

Denoting by $\Phi(x; \mu, \sigma) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ the normal distribution, the equation of the left hat kernel function is

$$H(x, m; a, b, \sigma_1, \sigma_2) = \begin{cases} r \Phi(x; m - b, \sigma_2) & \text{if } x \leq m - b \\ \frac{1-a}{b}(x - m) + 1 & \text{if } m - b < x < m \\ \Phi(x; m, \sigma_1) & \text{if } x \geq m \end{cases}$$

where a is the slope of the straight line, b is the distance from m at which the customer might have bought and $\sigma_i, i = 1, 2$ are the variances of the two normal distributions.

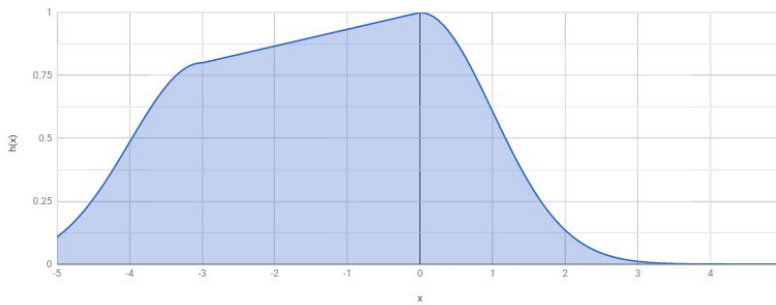


Figure 4.9: Left hat kernel function with $m = 0$, $a = 0.8$, $b = 3$, $\sigma_1 = 1$ and $\sigma_2 = 1$.

By combining these kernel marginal functions for each dimension of the state-action pair, it's possible to find the joint kernel function $k(x, x_e)$, which is used as attenuating factor when updating the value distributions.

4.3.2 Value distributions

The market response model associates to each state-action pair a value distribution that represents the distribution of the Q-value associated to that pair. This distribution is a density probability function for different values of Q-value. As seen before, the use of distribution is motivated by the fact that we want to have the most detailed information about the Q-values.

Let us consider a single state-action pair x and observe how we can build and update its value distribution. For each event, denote with x_e the state-action pair of the event, with v_e the Q-value observed and with w_e the weight of that observation. The weight of an observation is related to the cross-item information propagation component and its value will be explained better in the next section. The weight has value 1 for events relative to the item for which we are building the market model. Finally, let us denote with t the time at which there was the last update of the value distribution and with t_e the time at which the event occurs.

Unlike the Bayesian Q-learning algorithm, the value distribution adopted in Liquidprice could be of three types: the Beta Distribution, the Simple Value Distribution or the Count Distribution.

Binary reward signal

The Beta distribution² $\text{Beta}(a, b)$ is used in situations in which the Q-value should be a rate in $[0, 1]$ and the observations of rewards take value in $\{0, 1\}$ (bad or good reward). This could happen, for example, if the Q-value wants to model the conversion rate of a good (fraction of goods that are sold on the total number of offered goods).

Given an event (t_e, x_e, v_e, w_e) and the joint kernel function $k(x; x_e)$, the parameters are updated with the following rules, that take into account also the temporal decay coefficient d :

- case $t > t_0$: event older than last observed event

$$\text{if } v > 0 \quad \text{then} \quad a' = a + dw_e k(x; x_e) \quad \text{else} \quad b' = b + dw_e k(x; x_e)$$

- case $t \leq t_0$: event newer than last observed event

$$\text{if } v > 0 \quad \text{then} \quad a' = da + w_e k(x; x_e) \quad \text{else} \quad b' = db + w_e k(x; x_e)$$

In practice, the parameter a represents the mass of positive observations while b is the

²see[3] for details

mass of negative observations. Then the estimate μ of a Beta distribution is given by the fraction of positive observations over the total:

$$\mu = \frac{a}{a+b}$$

The theoretical precision of a Beta distribution is computed as

$$\tau = \frac{(a+b)^2(a+b+1)}{ab}$$

However, for the purpose needed here, we can simplify the concept of precision and approximate it with $\tau = a + b$, which is the total mass of information observed.

Continuous reward signal

If we want to estimate a continuous value in a wider range than $[0, 1]$ (as cases in which Q-value is computed as the mean of observed rewards), it's possible to assume that this value is normally distributed with unknown mean and precision. Then, we could use the Normal-Gamma distribution to estimate these unknown mean and variance, as described in [2] and [14].

However, the Normal-Gamma distribution has a weakness point: it's difficult to tune initial values for all the four hyperparameters and this increases the complexity of the problem. Hence, the Normal-Gamma distribution could be approximated by the Simple Value Distribution, that uses only two parameters that are directly the estimate μ and the precision τ : the estimate is a sort of mean of the observations, while the precision is the mass of observed information.

For each event (t_e, x_e, v_e, w_e) , given the joint kernel function $k(x; x_e)$ and the temporal decay coefficient d , estimate and precision of the Simple Value distribution are updated as follows

- case $t > t_0$: event older than last observed event

$$\tau' = \tau + d w_e k(x, x_e) \quad \mu' = \frac{\mu \tau + v_e d w_e k(x, x_e)}{\tau'}$$

- case $t \leq t_0$: event newer than last observed event

$$\tau' = d \tau + w_e k(x, x_e) \quad \mu' = \frac{\mu d \tau + v_e w_e k(x, x_e)}{\tau'}$$

Finally, the Count Distribution is used when we just want to store the mass of observations for a certain state-action pair. If we consider all the state-action pairs, this distribution works as a histogram.

The Count distribution has only one parameter n that collects the mass of observations and estimate and precision are simply given by $\mu = \tau = n$. For each event (t_e, x_e, v_e, w_e) , given the joint kernel function $k(x; x_e)$ and the decay coefficient d , the parameter n is update with the following rules:

- case $t > t_0$: event older than last observed event

$$n' = n + d v_e w_e k(x; x_e)$$

- case $t \leq t_0$: event newer than last observed event

$$n' = d n + v_e w_e k(x; x_e)$$

4.3.3 Market Models for insurances

In the problem of pricing insurances, we developed two possible market models among the many possible. However, in every model that can be built, the state-action pair consists only of a single variable, the flat value, that represents the action, since items do not have any state variable. Hence, the value distributions are two-dimensional (Q-values depend only on flat value) and the joint kernel function is just the kernel marginal function relative to the flat value dimension.

Unit Margin Market Model

The first model we built is called Unit Margin Market Model and tries to directly estimate the unit revenue obtained for each flat value by each item. For this purpose, we use the Simple Value Distribution with a Gaussian kernel. The use of Gaussian kernel is motivated by the fact that the margin received for a flat value influences the distribution of a small set of close flat values. The updating rules are the same as described above for the Simple Value Distribution, using as value of the event v_e the margin obtained on that event (it is 0 if there is not an insurance purchase).

Note also that this model really estimates the unit margin, since the information about the margin, contained in the market event, already considers possible costs and other indirect marginalities, such as financial or fiscal premium. The computation of the net margin is done by the selling systems of the merchant and allows Liquidprice to use it as a net unit margin.

Let us consider an item and simulate the behavior of the Unit Margin Market Model. We choose a Gaussian kernel with $\sigma = 2$: this choice implies that a large fraction of the weight observed by the flat value of the event would be observed by flat values that are far up to two units from x_e .

As first case, we choose $h = 20$ days as halving time parameter for the temporal decay

	t_0	x_0	w_0	m_0
#1	0.8	10.86	1	12.75
#2	8.7	14.44	1	0.00
#3	16.1	22.01	1	10.70
#4	25.6	15.60	1	13.89
#5	28.2	13.49	1	16.58
#6	37.3	19.88	1	0.00

Table 4.1: Market events - case 1.

coefficient and we select as initial market model the one obtained after observing three months of purchase data. After that time, we select a sequence of market events that are far from each other in time, in order to underline the effect of temporal decay and to emphasize the change in the model generated by each event. In real use case, events related to one item could be more frequent. Events data for this case are summarized in Table 4.1.

In Fig. 4.10 we can see the evolution of the market model after observing the different events. If the event generates a positive margin, for flat values near x_e , the estimate increases and the precision increases too, since there are more observations on that flat values. However, in the precision curve the major effect is given by the temporal decay, which decreases the all curve, due to passing of time. In the estimate curve, the effect of temporal decay is indirect since it comes from precision used for computing the weighted mean in the Simple Value Distribution.

Otherwise, when there is a negative event, the estimate decreases for flat values near to x_e , since the margin obtained with that price is null. Differently, the precision curve increases for flat values near x_e and decreases for temporal decay effect. In fact, the increasing behavior is due to the fact that we are increasing our knowledge about the performances of that flat values.

	t_0	x_0	w_0	m_0
#1	2.5	19.42	1	0.00
#2	3.1	14.54	1	15.04
#3	3.9	11.09	1	12.05
#4	5.8	23.39	1	0.00
#5	6.7	28.44	1	0.00
#6	7.8	10.77	1	0.00

Table 4.2: Market events - case 2.

For the second case, we choose $h = 30$ days as halving parameter for the temporal decay and the initial model is given by observing almost two months of events. In this situation, we choose an higher frequency of events, which is more similar to the real frequency. The market events observed are in Table 4.2.

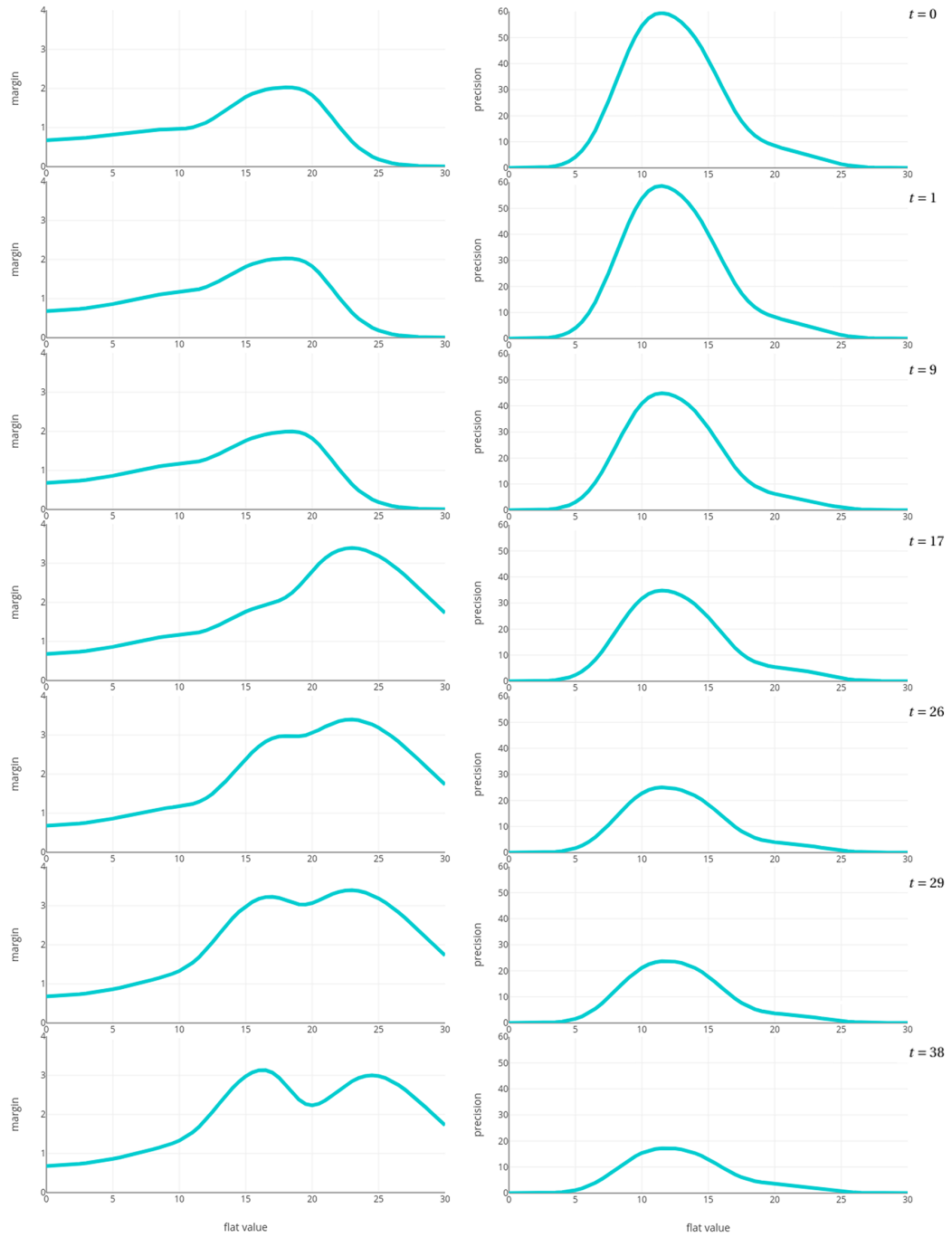


Figure 4.10: Estimate and Precision curve for Unit Margin Market Model - case 1.

In this case (Fig. 4.11) we observe that the effect of temporal decay is decreased. This is due to the fact that we have increased the halving time and decreased the time between two consecutive events. Also in this case we observe that the precision curve evolves always in the same way (it increases for flat values near x_e), since positive and negative events concur in a

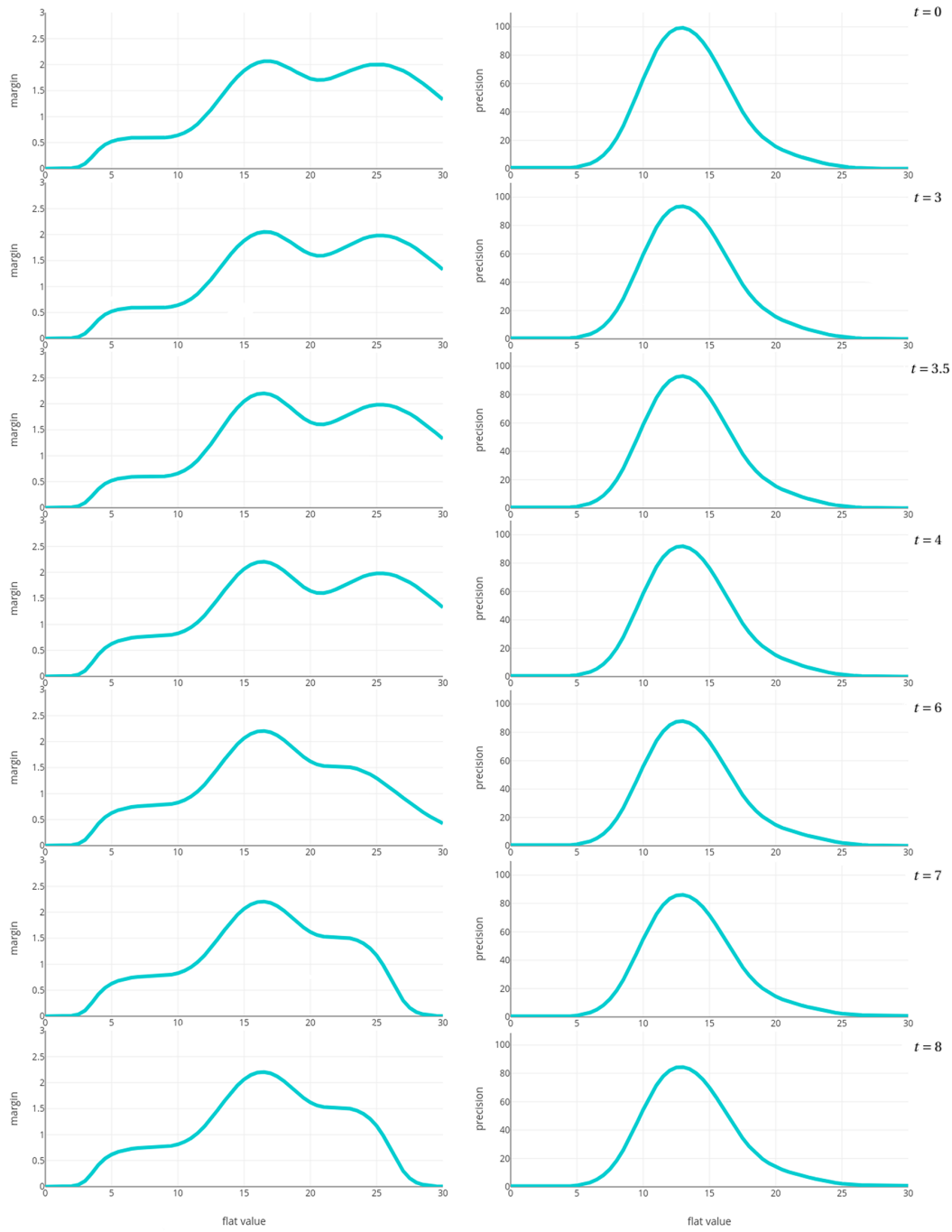


Figure 4.11: Estimate and Precision curve for Unit Margin Market Model - case 2.

similar way on the mass of information as before. Meanwhile, the estimate curve increases where the flat values have brought a positive margin and decreases where the flat values have led to a margin equal to zero.

Attach Rate Market Model

The other model we consider is called Attach Rate Market Model and predicts the insurance conversion rate (called *attach rate*) for each flat value and for each item. For this model we use the Beta Distribution, where the parameters a and b are the estimate of other two Count Distribution. In practice, the mass of positive observations and the mass of negative observations are stored as distribution and their estimates are used for building the final Beta Distribution.

Therefore, events where there is an insurance purchase would update the positive Count Distribution $p(x)$ with $(v_e = 1, w_e = 1)$ and the negative Count Distribution $n(x)$ with $(v_e = 1, w_e = 0)$, in order to consider the effect of temporal decay. Similarly, events without insurance purchases would update the negative Count Distribution $n(x)$ with $(v_e = 1, w_e = 1)$ and the positive Count Distribution $p(x)$ with $(v_e = 1, w_e = 0)$.

The kernel functions for these Count Distributions represent the willingness to pay of customers and they can be Sigmoid Higher Floor kernel functions. The use of the Higher Floor function is motivated by the fact that there are customers like organized travels that would buy an insurance at any cost and there are other types of customers that don't want to buy an insurance, whatever the cost is. Then, we choose a decreasing Sigmoid kernel function for $p(x)$, while $n(x)$ is its symmetric function with respect to the vertical axis (i.e. $n(x) = p(-x)$). In this way, $n(x)$ is an increasing function with the same parameters of $p(x)$. The only parameter that distinguishes the two kernel is the parameter of the Higher Floor function f , since the probability of buying in any case could be different from probability of never buying.

Finally, the parameters of the Beta distribution are $a(x) = \mu_p(x)$ and $b(x) = \mu_n(x)$. Since $\mu_p(x)$ and $\mu_n(x)$ are the estimates of two Count Distribution, the parameters a and b are always positive, as required by the Beta distribution. Hence, the precision of the Beta distribution is always positive, since it is the sum of two positive quantities:

$$\tau(x) = a(x) + b(x) = \mu_p(x) + \mu_n(x)$$

Moreover, due to the choice of the kernel functions, the positive estimator is a decreasing function and the negative estimator is an increasing function. Then, the parameter $a(x)$ is decreasing, while $b(x)$ is increasing. With this information, we can study the monotony of the attach rate estimate function. In fact, the attach rate should be a decreasing function, because it should modeled how many people would buy an insurance at a certain price. This is a consequence of the willingness to pay of customers: the number of customers that would buy at a low price is higher than the number of customers that would buy at a high price.

Let us consider the function that determines the estimate of the attach rate:

$$\mu(x) = \frac{a(x)}{a(x) + b(x)}$$

In order to find the monotony, we consider the first order derivative of the estimate function with respect to the flat value x :

$$\begin{aligned}\mu'(x) &= \frac{a'(x)[a(x) + b(x)] - a(x)[a'(x) + b'(x)]}{[a(x) + b(x)]^2} \\ &= \frac{a'(x)b(x) - a(x)b'(x)}{[a(x) + b(x)]^2}\end{aligned}$$

Then, recalling that $a'(x) < 0$ and $b'(x) > 0$, we have that $\mu'(x) < 0, \forall x$. Therefore, the attach rate is a decreasing function, as expected.

In order to check these properties of the Attach Rate Market Model, let us consider an item and see the evolution of the model. Before observing the results obtained by using the Higher Floor Sigmoid kernel, let us see how is the resulting market model using the Sigmoid kernel function. This would explain why we need to use the composition with the Higher Floor kernel function.

	t_0	x_0	w_0	m_0
#1	0.8	10.86	1	12.75
#2	8.7	14.44	1	0.00
#3	16.1	22.01	1	10.70
#4	25.6	15.60	1	13.89
#5	28.2	13.49	1	16.58
#6	37.3	19.88	1	0.00

Table 4.3: Market events - case 1.

Let us consider two Sigmoid kernels for positive and negative estimators with same parameters $b = 1$ and $c = 0.99$. The initial model is obtained by observing three months of data and we choose a sequence of market events far each other over time. The temporal decay coefficient has parameter $h = 20$ days as halving time. The market events are in Table 4.3.

In Fig. 4.12 there are depicted the changes due to market events on positive and negative estimators. In particular, when there is a purchase, the positive estimator is updated with an observation of weight 1, while the negative estimator is updated with an observation of weight 0, in order to consider only the effect of temporal decay. In fact, the two estimators must be synchronized in terms of time of last update.

On the other hand, when there is a negative observation, the positive estimator receives an event with weight 0 and the negative estimator receives an event with weight 1. Then, when an estimator is updated with weight 1, it increases, while it decreases for the effect of temporal decay or when it is updated with an event of weight 0.

The final behavior of the market model is depicted in 4.13. When there is a positive events, the estimate curve increases, while it decreases when there are negative events and for the indirect effect of temporal decay. Moreover, neglecting the temporal decay effect, the precision

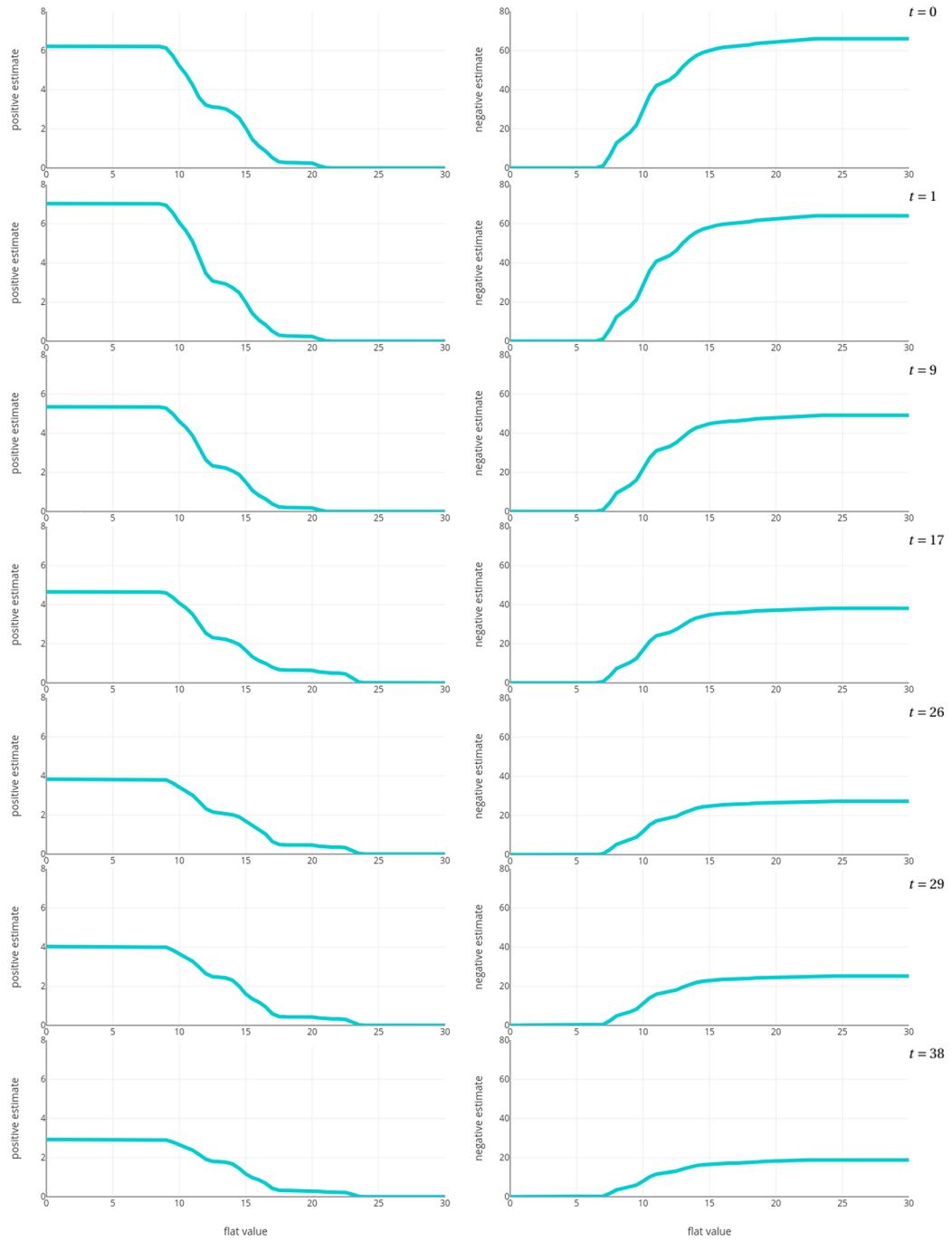


Figure 4.12: Positive and Negative estimator for Attach Rate Market Model with Sigmoid kernel function - case 1.

curve increases every time we have an observation, since we are increasing our knowledge. However, in both curve, the negative estimator has more weight than the positive estimator, since it takes higher values, due to the presence of more negative than positive observations in the data observed.

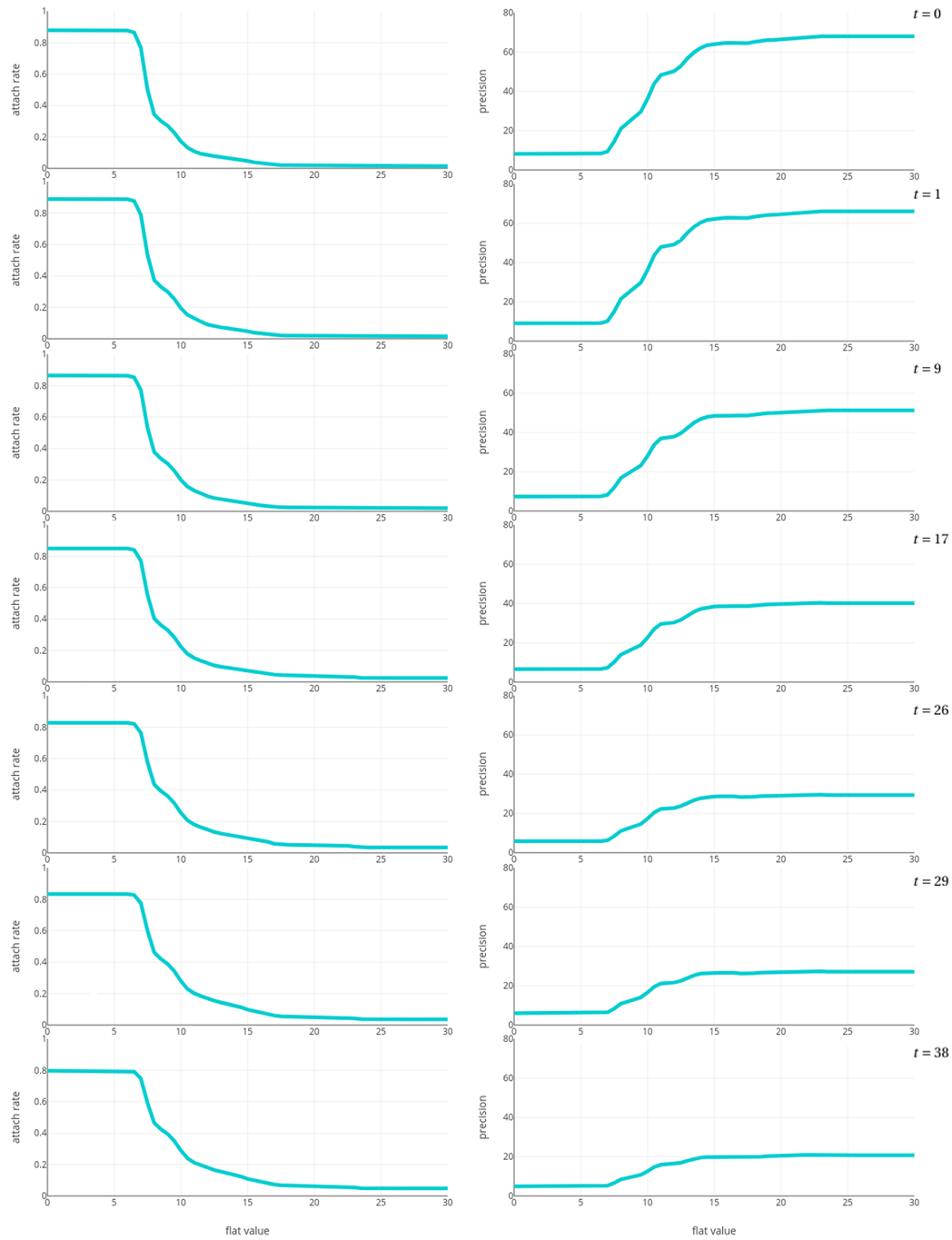


Figure 4.13: Estimate and Precision curve for Attach Rate Market Model with Sigmoid kernel function - case 1.

However, the use of sigmoid kernel has a non negligible drawback: the resulting models are very overbalanced on left and right tails (very low and very high flat values). This leads to overestimate the attach rate for low flat values and underestimate it for high values. This fact motivates the use of the Higher Floor Sigmoid kernel functions for the estimators, instead of

the Sigmoid kernel.

Then, let us see the behavior of the Attach Rate Market Model using the Sigmoid Higher Floor kernel for both positive and negative estimators. As before we choose $b = 1$ and $c = 0.99$ as parameter of the sigmoid functions, while for the higher floor parameters we estimate from historical data $f = 0.1$ for the positive kernel and $f = 0.5$ for the negative kernel.

Let us consider the sequence of market events in Table 4.3 and use as halving time parameter $h = 20$. Also in this case, we start with a model built on purchase data observed in three months.

In Fig. 4.14 there are the evolution of the two estimators. As seen before, positive events would update with weight 1 the positive estimator and with weight 0 the negative estimator, in order to consider the temporal decay. Meanwhile, negative events would update with weight 1 the negative estimator and with weight 0 the positive estimator. Moreover, we could observe that these estimators have fat tails, due to the effect of the Higher Floor function.

The final Attach Rate Market Model (in Fig. 4.15) would take into account the information stored in the two estimators. As proved above, the estimate curve is a decreasing function. Furthermore, note that the precision curve has a decreasing evolution due to the temporal decay effect. Also in this case, the weight of the negative estimator in the final model is higher than the weight of the positive estimator. This is due to the fact that there are more negative than positive events.

Moreover, in contrast with the previous model with the use of the Sigmoid kernel, the estimate of the attach rate in this model is not affected by an overestimate, thanks to the effect of the Higher Floor Sigmoid kernel function.

	t_0	x_0	w_0	m_0
#1	2.5	19.42	1	0.00
#2	3.1	14.54	1	15.04
#3	3.9	11.09	1	12.05
#4	5.8	23.39	1	0.00
#5	6.7	28.44	1	0.00
#6	7.8	10.77	1	0.00

Table 4.4: Market events - case 2.

Finally, let us consider another Attach Rate Market Model, with same parameters as previous one, but using another set of events that are more frequent over time and using $h = 30$ days as halving time parameter for the temporal decay coefficient. In this way, we want to decrease the effect of temporal decay in order to observe better the changes in the model due to the market events. The initial model is built using two months of purchase data and the market events are summarized in Table 4.4.

The resulting positive and negative estimators for this set of events are depicted in Fig.4.16.

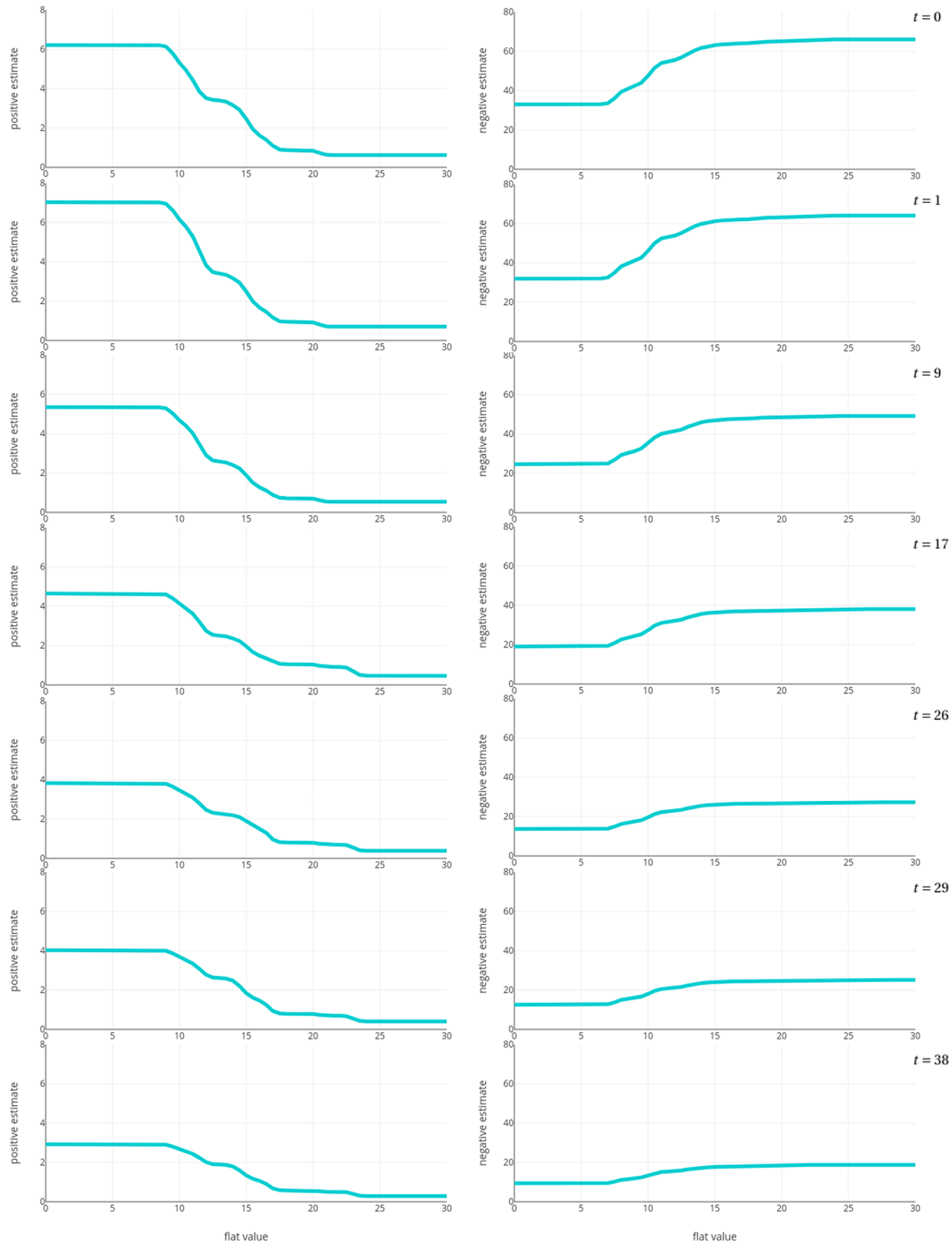


Figure 4.14: Positive and Negative estimator for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 1.

For the events choice and for the choice of halving time, the effect of temporal decay is decreased. Then it's possible to see that the positive estimator increases when there is a purchase, while the negative estimator increases when there are negative events.

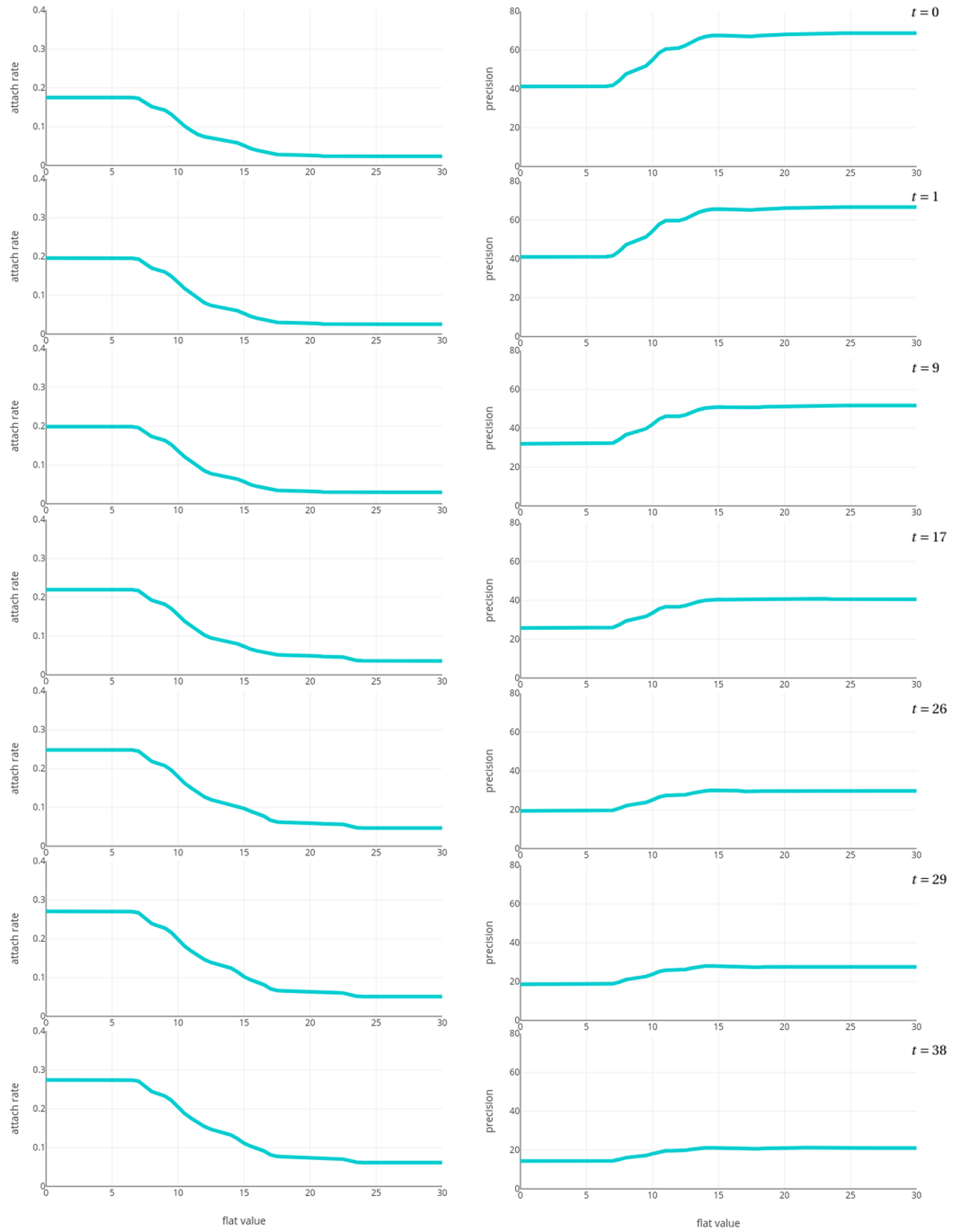


Figure 4.15: Estimate and Precision curve for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 1.

In Fig. 4.17 there is the resulting Attach Rate Market model. Also in this case, we could see that the attach rate is not overestimated for low flat values and it is not underestimated for high flat values.

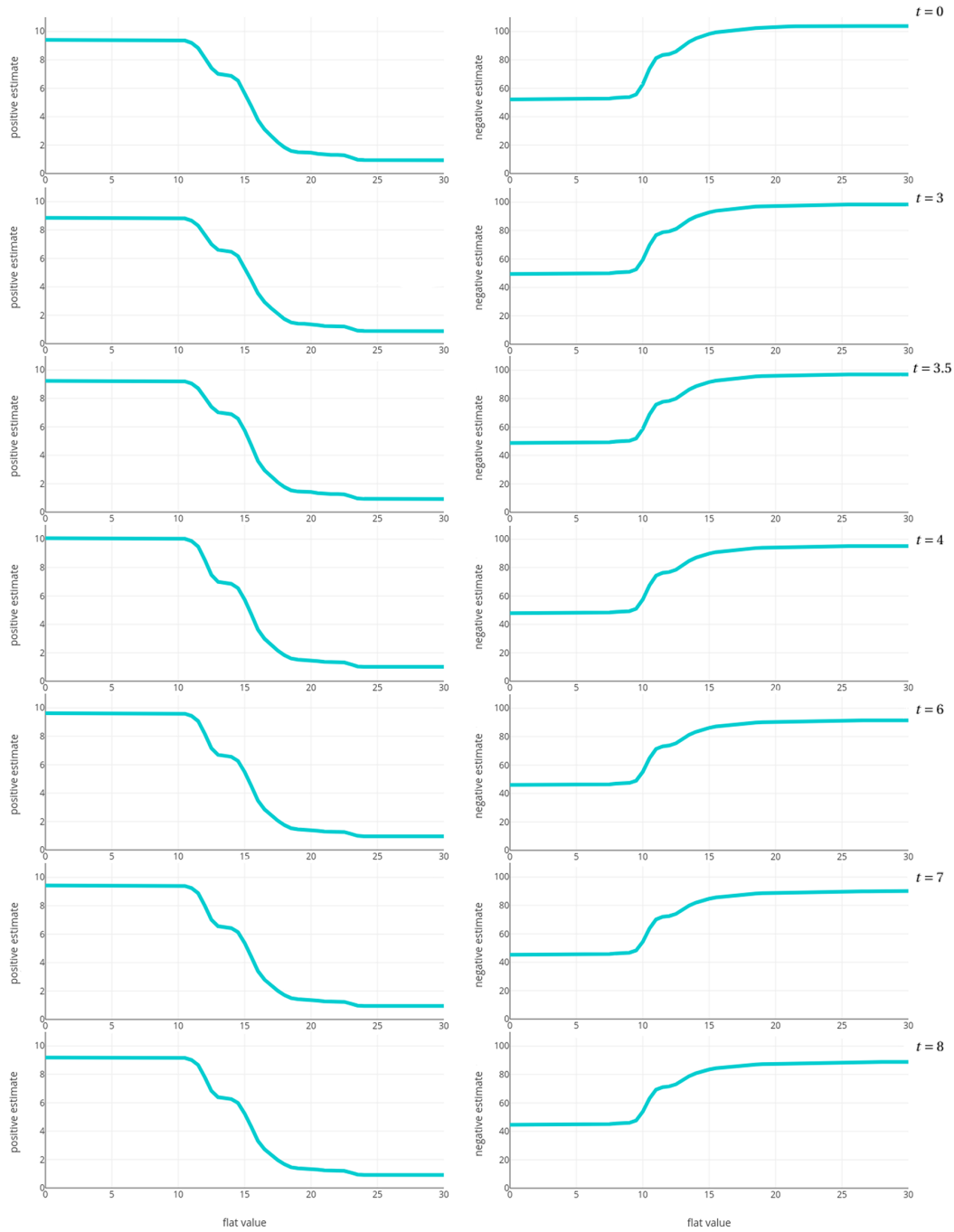


Figure 4.16: Positive and Negative estimator for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 2.

Finally, although the Attach Rate Market Model allows more reasoning about conversion rate and customer behavior than what we can do using the Unit Margin Market Model, we think that for the context of insurance pricing the Unit Margin Market Model is more promising

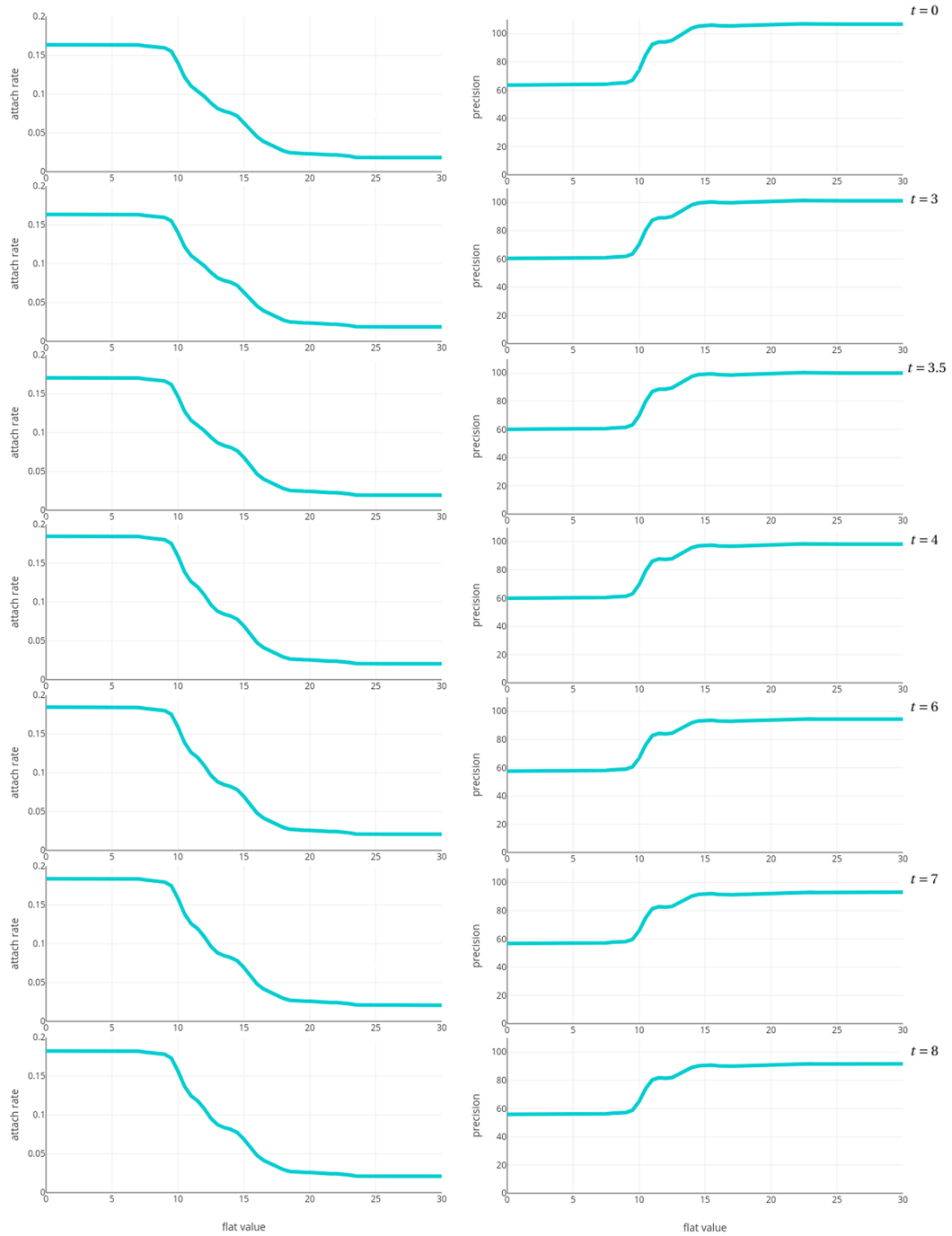


Figure 4.17: Estimate and Precision curve for Attach Rate Market Model with Higher Floor Sigmoid kernel function - case 2.

and gives a better explanation of market response. This is also due to the fact that it uses the real net unit margin, coming directly from the selling systems of the merchant.

4.3.4 Market Models in contexts without negative observations


Previously, we have seen that in some contexts not all the purchase information are available. For example, some companies do not have data about customers that look for a product but don't buy it (negative market events). Liquidprice tries to overcome this lack of information by adding dummy market events that represent the missing negative events.

However, it's not so obvious how to create these dummy events. In fact, there are two different situations to be considered: adding dummy events for historical data and adding dummy events while the software is being used.

The historical data are used to create the initial market model. For these data, we have no information about the price proposed to customers. It's reasonable to think that before using Liquidprice, the company would have used a static pricing strategy. Hence, it's possible to assume that the price for which we observed a purchase was the price proposed until that moment.

With this assumption, a possible solution is to penalize prices that have been posted for a long time but have not lead to a purchase. Then, we choose a unit time measure (such as one day or one hour) and we let a clock start at time 0. Every time the clock strikes, the system will add a negative event. The dummy event has the same information of the first purchase event that occurs after the time of the dummy event. In this way, the dummy event has the price of the subsequent purchase.

In order to see how this process works, we try to apply it in the insurance context, assuming that we have only positive market events as historical data. In Fig. 4.18 there is a possible outcome of the addition of dummy events, assuming unit time measure equals to 1.5.



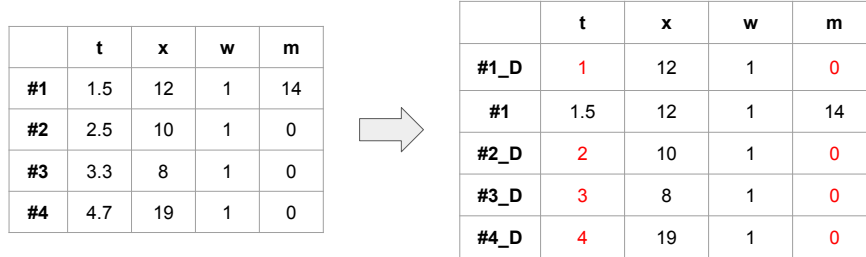
	t	x	w	m
#1	1	12	1	14
#2	4	8	1	10
#3	5	19	1	16

	t	x	w	m
#1	1	12	1	14
#1_D	1.5	8	1	0
#2_D	3	8	1	0
#2	4	8	1	10
#3_D	4.5	19	1	0
#3	5	19	1	16

Figure 4.18: Addition of dummy market events for historical data.

Differently, when Liquidprice is active, the company is using a dynamic pricing strategy, hence prices could change very frequently. However, the software keeps track of the prices proposed at each time instant. Then, choosing as unit time measure the time interval for which a price is proposed, we could add a negative event with that price, every time the clock strikes.

Using the context of insurance pricing, we could see a practical application of this process. In Fig. 4.19 it's shown how we try to replicate the real negative events with dummy ones.



	t	x	w	m
#1	1.5	12	1	14
#2	2.5	10	1	0
#3	3.3	8	1	0
#4	4.7	19	1	0

	t	x	w	m
#1_D	1	12	1	0
#1	1.5	12	1	14
#2_D	2	10	1	0
#3_D	3	8	1	0
#4_D	4	19	1	0

Figure 4.19: Addition of dummy market events.

We will see how the market model changes only in the case of the Unit Margin Market Model since the Attach Rate Market Model assumes to have negative events. Adding them as dummy events, it makes no sense to estimate the attach rate, because it is defined as fraction of positive events over total events.

Unit Margin Market Model with dummy events

Let us consider a Unit Margin Market Model with a Gaussian kernel ($\sigma = 2$) and with halving time $h = 20$ days for the temporal decay coefficient. Clearly, the estimate of the model obtained using dummy negative events is not a real estimate of the margin, but just a margin indicator, that is an indicator of prices performances.

First of all, we consider the addition of dummy market events in the case of historical data. We select 5 days of real historical data (Table 4.5) and the corresponding positive events with dummy negative events (Table 4.6). The unit time measure is one day since in the insurance context we update prices once a day.

	t_0	x_0	w_0	m_0
#1	0.1	8.75	1	0.00
#2	0.6	8.75	1	0.00
#3	1.3	14.00	1	0.00
#4	2.1	14.00	1	0.00
#5	3.1	9.75	1	0.00
#6	3.2	9.75	1	0.00
#7	3.4	9.75	1	0.00
#8	3.5	9.75	1	0.00
#9	3.6	9.75	1	0.00
#10	4.4	13.00	1	0.00
#11	4.5	13.00	1	14.34

Table 4.5: Real market events for 5 days of historical data.

	t_0	x_0	w_0	m_0
#1_D	0.1	13.00	1	0.00
#2_D	1.1	13.00	1	0.00
#3_D	2.1	13.00	1	0.00
#4_D	3.1	13.00	1	0.00
#5_D	4.1	13.00	1	0.00
#1	4.5	13.00	1	14.34

Table 4.6: Real positive market events with dummy negative events for 5 days of historical data.

Chapter 4. Adapting reinforcement learning to dynamic pricing: Liquidprice

Starting with a market model representing market events of past four months, it's possible to compare the two models at the end of each day. First of all, we observe that the set of real events has a large variety of prices for which we have the market response. On the other hand, the set with dummy events gives only a partial information about the market response, since we could give a good or bad reward only to a small set of prices (in this case only to one price, since we have one positive market event).

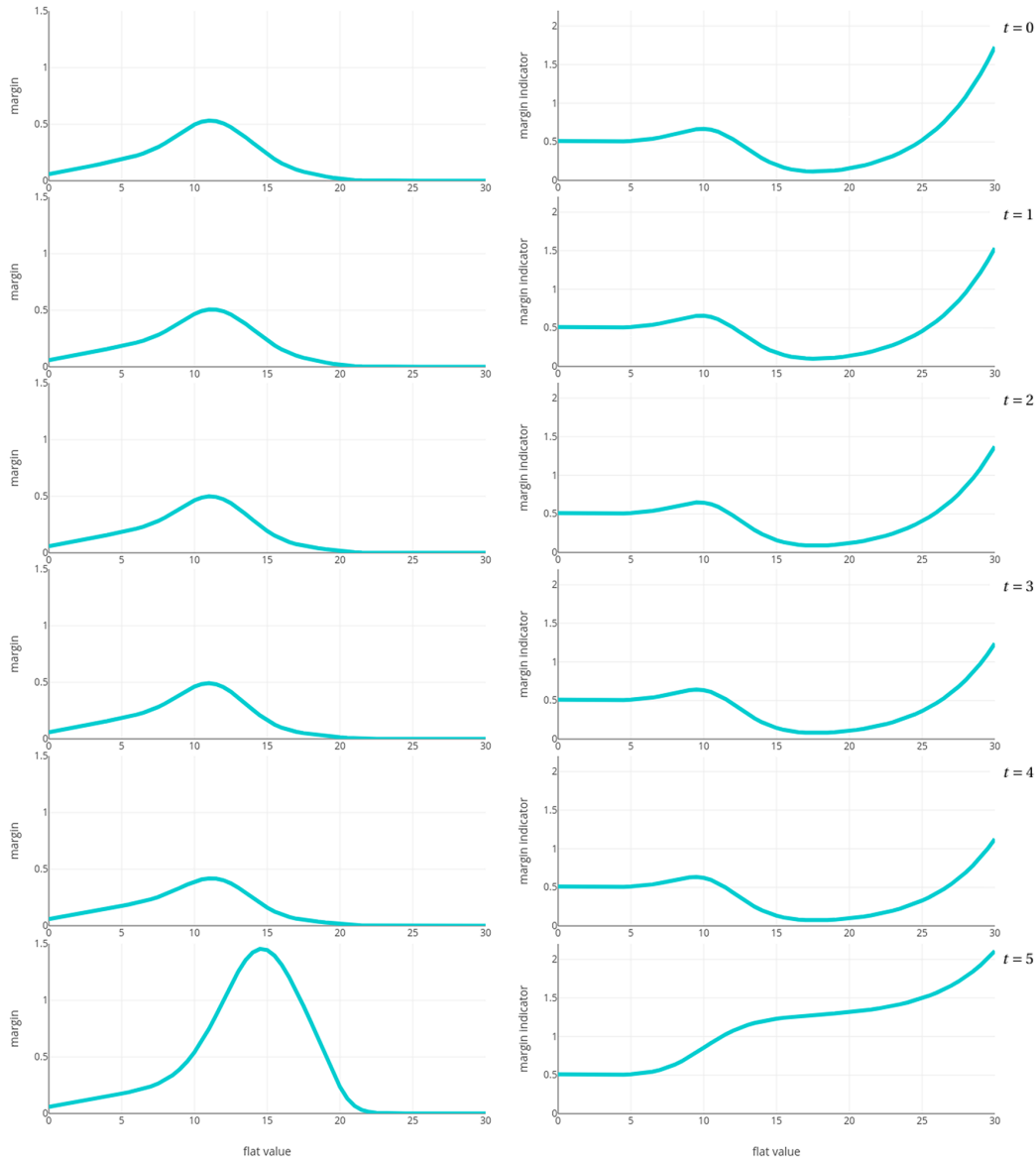


Figure 4.20: Historical dummy events - comparison between estimate curves.

This partial information is well represented in Fig. 4.20 by the comparison of estimate curves. For flat values between 5 and 15, the estimate of the two curves are similar. On the other hand, for very low and very high prices, the estimate of the model with dummy events

has an unexpected behavior. This is due to the fact that for those flat values the precision is low (close to 0), then due to the updating rule of the Simple Value Distribution, the resulting estimate is very large. This behavior is not observed in the real market model since we have more information for different prices.

The problem of these overestimates due to low precision can be solved by choosing a suitable pricing policy, as it will be explained in the next section.

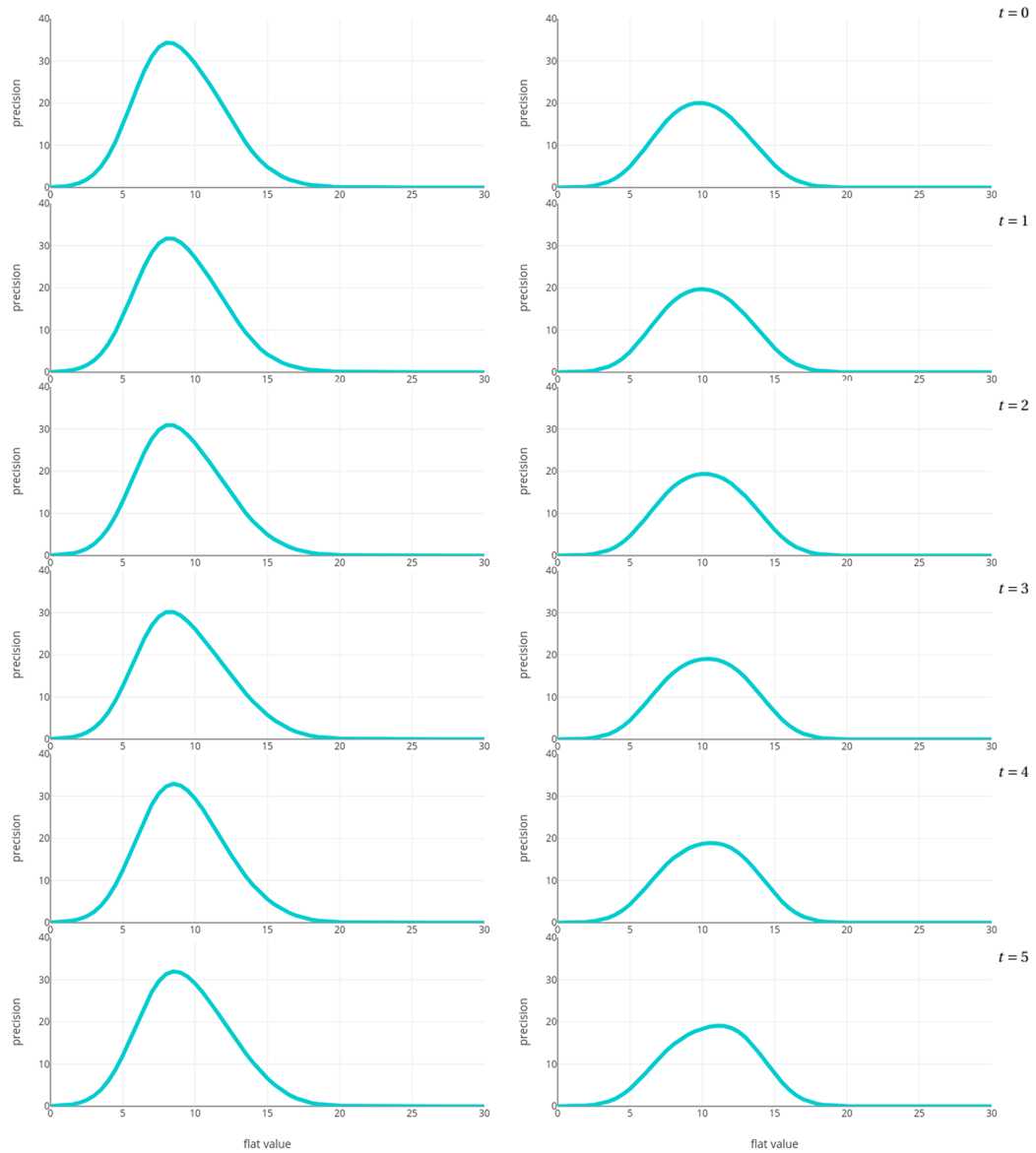


Figure 4.21: Historical dummy events - comparison between precision curves.

The different mass of information could also be observed in Fig. 4.21. We note that the real market model has a larger precision than the model with dummy events. This is due to the

fact that in the dummy events we only have information about prices that lead to a purchase, while we don't have any information about other possible prices that can have been posted without leading to a purchase. In fact, the addition of dummy market events is just a way to give a weight to good prices, assuming that they were the only prices proposed.

This weight is given by the time that passes between two events: if it is short, that price obtained a very good reward with a high weight (there are few dummy market events added); otherwise, if that time is long, there are more dummy negative events that decrease the weight given to the flat value of the positive observation.

Finally, in these two models, the effect of temporal decay is minimized, since we are considering events that are very close one another, hence the temporal decay coefficient applies a higher rescaling factor (almost 1), which means that there is a very low decay.

The second situation is the addition of dummy events while using the dynamic pricing software. For this situation we consider market events of 6 days in Table 4.7. On the other hand, in Table 4.8 we have the real positive events with the addition of the negative dummy events with the price proposed on each day. Even in this case, the unit time measure is one day.

	t_0	x_0	w_0	m_0
#1	0.4	17.50	1	0.00
#2	0.9	17.50	1	0.00
#3	2.4	11.75	1	0.00
#4	4.5	9.50	1	0.00
#5	5.2	13.11	1	8.64

Table 4.7: Real market events for 5 days of dynamic pricing data.

	t_0	x_0	w_0	m_0
#1_D	0.1	17.50	1	0.00
#2_D	1.1	11.75	1	0.00
#3_D	2.1	11.75	1	0.00
#4_D	3.1	9.50	1	0.00
#5_D	4.1	9.50	1	0.00
#6_D	5.1	13.11	1	0.00
#1	5.2	13.11	1	8.64

Table 4.8: Real positive market events with dummy negative events for 5 days of dynamic pricing data.

Starting with a model obtained by the process described above with four months of historical data, we could compare the evolution of the Unit Margin Market Model at the end of each day. In this case, we have a large variety of different prices that are observed. Then we could think that the two models would be similar.

In Fig. 4.22 there is the comparison of the two estimate curves. We observe that the estimate of the margin is similar for central flat values. Meanwhile, the estimate for low flat values in the model with dummy events is higher than the corresponding estimate in the real model. As before, this is an effect due to low precision that could be overcome with suitable pricing policy.

Observing the precision curves in Fig. 4.23, we see again that the model built with dummy events has lower precision. In general, indeed, it observes fewer market events than the real

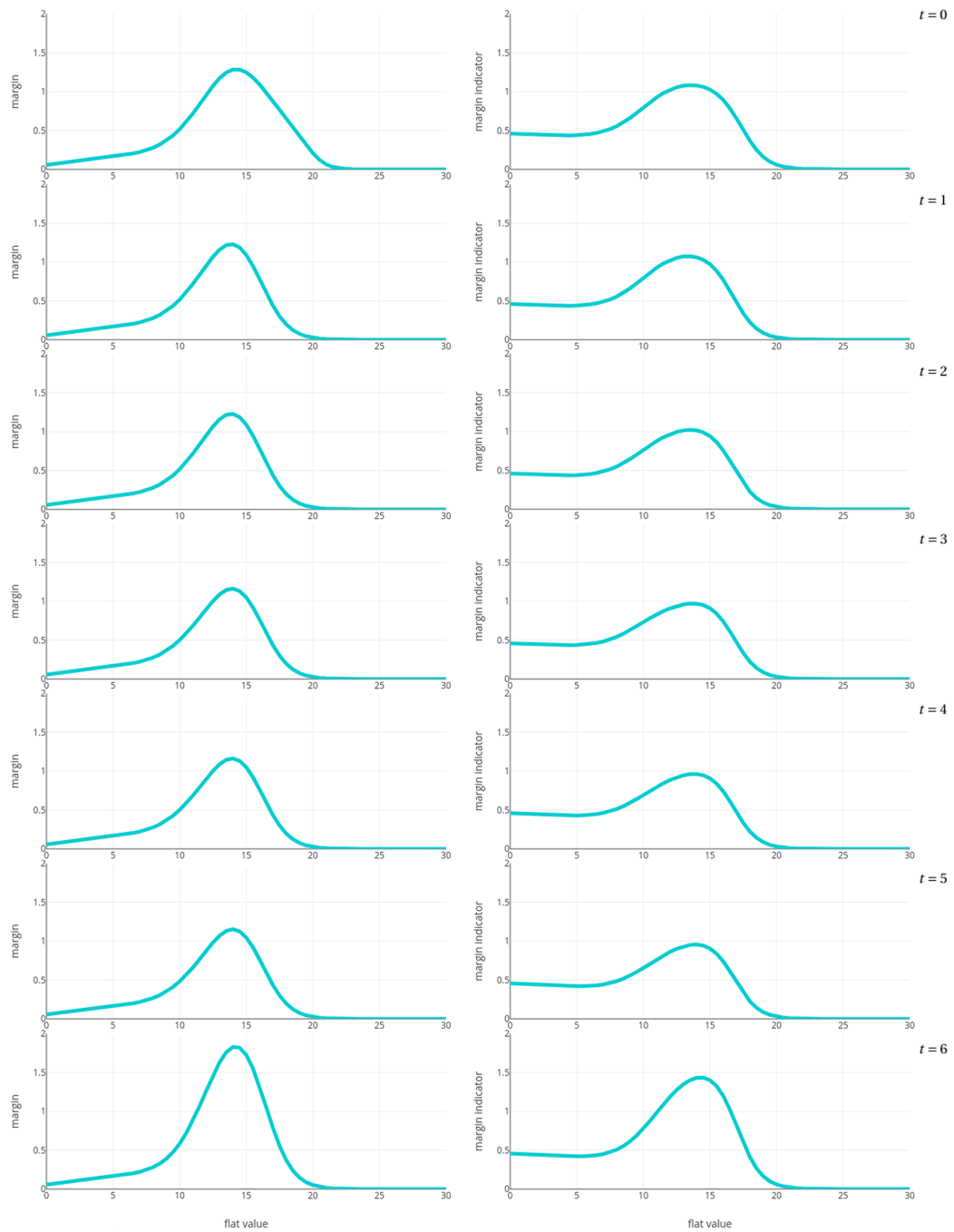


Figure 4.22: Dummy events - comparison between estimate curves.

model.

Finally, in contexts where we don't have data of negative market events, we could build market models using these two steps. By the comparison of the two models, we observe that

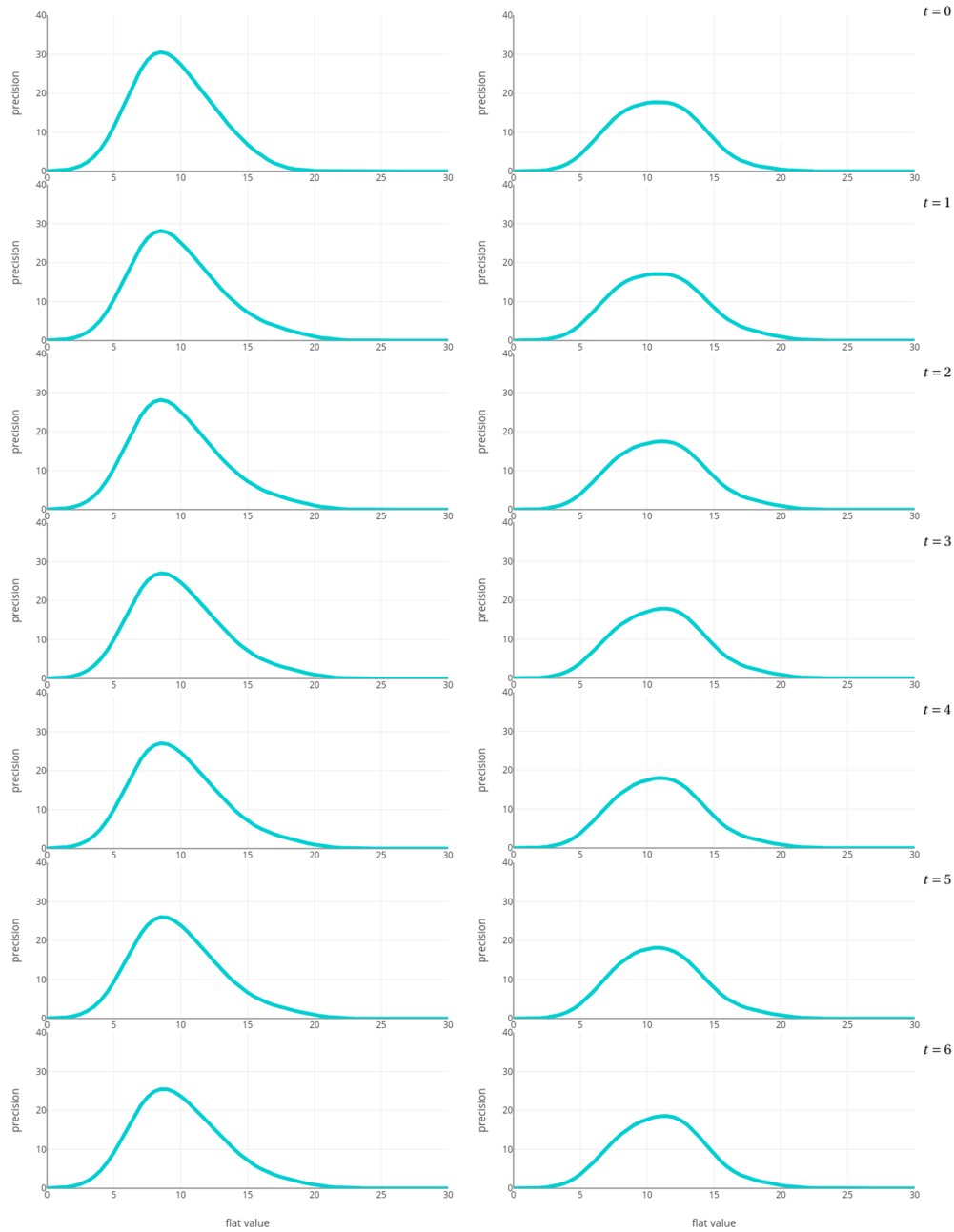


Figure 4.23: Dummy events - comparison between precision curves.

the model built with dummy events is a good approximation of the real market model.

In fact, the "dummy" model is slower to learn the real market model, but as time passes they get similar. Moreover, the maximum values' location is preserved, probably leading to similar pricing decisions depending on the policy used.

4.4 Pricing policy

The pricing policy is the component that takes the pricing decision, that is the selection of the new action that the agent performs. Indeed, the pricing decision is the final phase of the pricing mechanism and it is based on the market model and on some business-driven parameters and rules that depend on the contexts of application.

As in any reinforcement learning problem, the pricing policy has to take into account the exploration-exploitation trade-off. Specifically, the policy has to be able to perform two kinds of actions:

- Explore the market behavior in order to assess the response to different prices.
- Exploit promising prices in order to optimize the objective function. These are actually the actions that allow achieving the objective of the company in their dynamic pricing problem.

Therefore, the pricing decision could follow any sort of policy that is able to perform these types of actions. Moreover, since Liquidprice uses a solution algorithm based on Q-learning, the policy that it uses are derived from the market model available at the moment of the pricing decision.

In practice, the policy takes as input the market model and consists of two main steps:

- derivation of the objective function $OF(x)$ from Q-values of market models: every function of the Q-values could be used.
- selection of the action, based on the objective function defined before: it could be deterministic or stochastic.

For example, in the ϵ -greedy method based on Q-values, the objective function is given by the Q-values, while the action selection is deterministic if we are exploiting and stochastic if we are exploring.

4.4.1 Pricing policies for insurances

We will present now some possible pricing policies for the context of flight insurances. The policies are based on the Unit Margin Market Model or on the Attach Rate Market Model described in a previous section.

Since the problem is a multi-armed bandit, we can use every policy described in Section 3.2 or every their evolution.

Moreover, recall that in this application the objective is to maximize revenue coming from insurance purchases.

Margin-based ϵ -greedy policy

The simplest policy that we can use is the ϵ -greedy policy, in order to maximize the unit margin.

Based on the Unit Margin Market Model available at the time of the decision, this policy aims to maximize the expected margin. Hence, the objective function is just the value of the margin, $OF(x) = m(x)$.

In exploitation mode, the policy selects, in a deterministic way, the flat value that maximizes the revenue. Since we want to be confident on the value of the expected margin, we consider as selectable actions only the flat values with an associated precision which is at least a fraction of the maximum precision observed: $\tau(x) \geq \gamma \max_i \tau(x_i)$. This is motivated by the fact that in exploitation we want to select actions for which we are confident about their goodness.

In exploration mode, instead of selecting randomly an action (the classical approach), we randomly select a flat value with a probability which is inversely proportional to the respective precision. In practice, we associate to each flat value a probability of being extracted $\pi(x)$ which is computed as

$$\pi(x) = \frac{\frac{1}{\tau(x)}}{\sum_k \left(\frac{1}{\tau(k)} \right)}$$

Then, we extract a flat value according to this probability, which encourage the selection of flat values with a low precision.

This policy could be improved by allowing a random selection also in exploitation actions. This is useful if there are many flat values that lead to the same (or very similar) margin and we don't want to select always the same action. Hence, we can assign to each selectable flat value a probability of selection which is proportional to the relative expected margin $\pi(x) = m(x)$. Then, we select with higher probability actions with higher margin, but, with a small probability, we could also select other flat values, performing a quite exploring action.

In Fig. 4.24 we have the estimate and the precision curve relative to a Unit Margin Market Model. From the precision curve, we observe that the selectable actions are the set of flat values between 7 and 21 since their precision is above the chosen minimum (with $\gamma = 0.1$). Note that the flat values under the minimum precision have the higher probability of being selected as exploration actions.

Moreover, we observe that if we select an exploitation action in a deterministic way, the policy would select a flat value around 16, where the margin is maximum. On the other hand, if we use a stochastic selection, the policy will select with higher probability the flat values between 12 and 19.

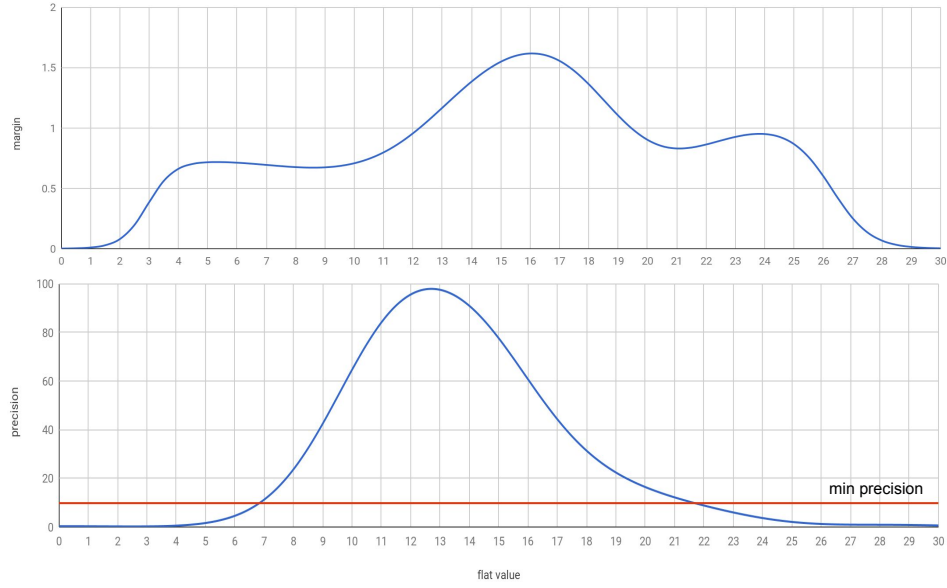


Figure 4.24: Estimate curve and Precision curve for a Unit Margin Market Model.

Attach Rate-based ϵ -greedy policy

An ϵ -greedy policy could be implemented also using the Attach Rate Market Model. In this case, we could not maximize directly the attach rate, but we should derive an expected margin indicator from the attach rate and then maximize that indicator. This is motivated by the fact that the company's objective is to maximize the margin obtained by selling insurances. In fact, if we maximize just the attach rate, we end up with a policy that increases the number of sold insurances and not necessarily the revenue. In fact, sometimes selling a large number of insurances at a low price could result in less revenue than what we can obtain by selling few insurances but at a larger price.

Hence, we should find a way for computing the expected margin for each flat value. Recall that the flat value is given by a percentage of the total cost of the flight. Let us assume that for each insurance that we sell we have a cost i_c which is a percentage of the total cost of the flight too. Denote with c the average of the total flight cost observed by an item. Since the item is characterized by the range of total flight cost, the average is a good estimator for computing the expected margin. Therefore, if $a(x)$ is the attach rate for the flat value x , the objective function is

$$OF(x) = \mathbb{E}[m(x)] = a(x) * c * (x - i_c)$$

Then, the exploitation mode selects the flat value that maximizes this expected margin. The action selection could be deterministic or stochastic in the same way as it is described for the Margin-based ϵ -greedy policy. The same holds for the selection of the exploration actions.

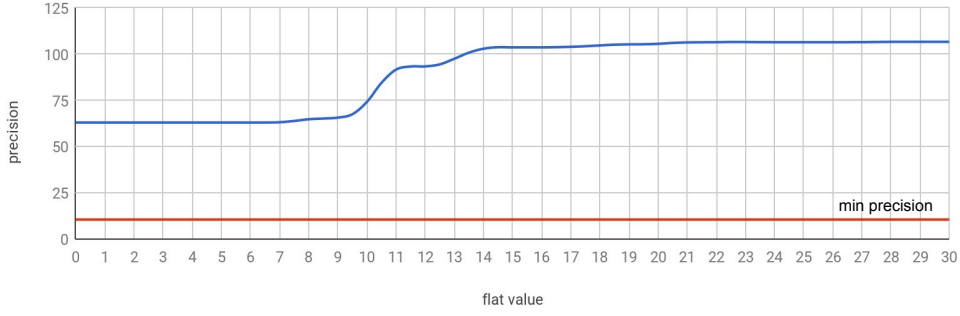


Figure 4.25: Precision curve for an Attach Rate Market Model.

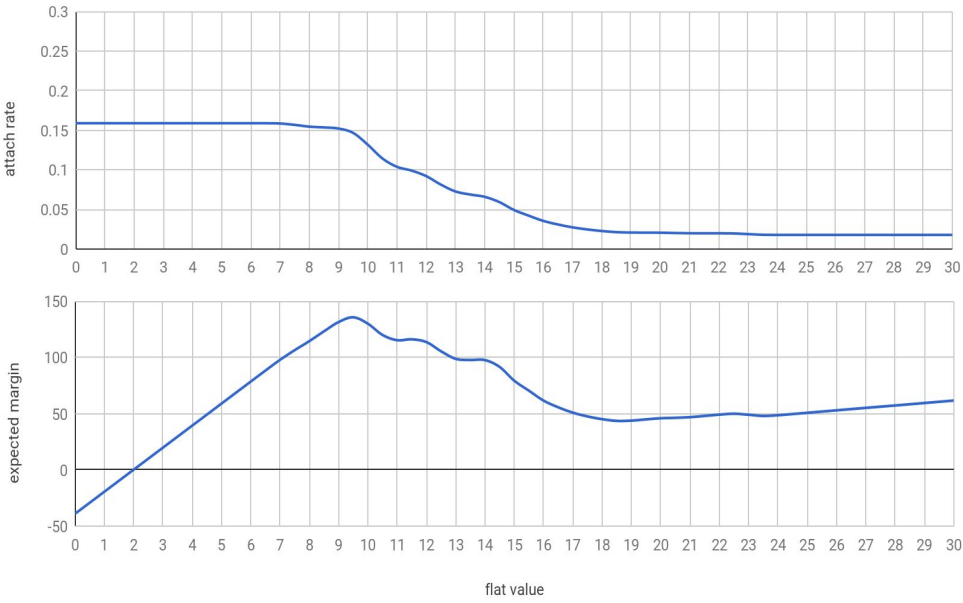


Figure 4.26: Comparison between Estimate curve and Expected Margin curve for an Attach Rate Market Model.

Choosing an Attach Rate Market Model for an item with $c = 123.44$ and $i_c = 2$, we obtain the precision curve in Fig. 4.25 and the objective function in Fig. 4.26. Observing the precision curve, we have that all the flat value are selectable actions, since they are all above the minimum with $\gamma = 0.1$. Moreover, the exploration actions would select with higher probability the flat values lower than 10.

From the comparison of the attach rate curve and the objective function, we could see that if we maximize directly the attach rate, we would select a flat value less than 7, while maximizing the expected margin, we select a flat value between 9 and 10. Hence, if we want to maximize the incomes and not the number of insurances sold, we should maximize the objective function, instead of the attach rate.

Note also that the revenue obtained for low flat values is negative and this is due to the cost i_c of selling an insurance. Actually, depending on the amount of this cost and for other business reasons, the flat value could not be less than 5%, hence situations with negative expected margin are avoided.

Margin-based Upper Confidence Bound Policy

As seen in section 3.2, a good policy that solves the exploration-exploitation trade-off is the Upper Confidence Bound Policy. We choose to apply this policy to the Unit Margin Market Model.

From Eq. 3.4, we know that the Upper Confidence Bound optimize a weighted sum of the reward estimate and the inverse of the square root of the precision (the mass of observations). Hence, this policy improves the exploration by selecting actions with high uncertainty (low precision) because this means that they have a high potential of being optimal.

Therefore, in order to compute the objective function based on the Upper Confidence Bound Policy for the insurance contexts, we should modify the theoretical function in an appropriate way. First of all, the logarithmic term related to time could be considered equal to 1, since the effect of time is already considered into the temporal decay coefficient used for building the market model.

The term dependent on time leads also to more exploration in the first steps of the algorithm and less when the optimum is being achieved. However, in dynamic pricing, the environment is non-stationary and there isn't a fixed optimal value to achieve. Hence, we could neglect the logarithmic term since it's necessary to have always a quite high degree of exploration in order to follow changes in the market and find the optimal price in each time period.

Moreover, in order to compute the parameter c representing how the precision is worth with respect to the estimate, precision and estimate should be comparable. For this purpose, denoting by x the flat value, we divide estimate $m(x)$ and precision $\tau(x)$ by their maximum values:

$$M = \max_x m(x) \quad P = \max_x \tau(x)$$

Then, as an example, we choose $c = 0.5$ which means that the estimate has double value with respect to the square root of the inverse of the precision. Hence, the objective function derived from the Unit Margin estimate is

$$OF(x) = \frac{m(x)}{M} + c \sqrt{\frac{1}{\tau(x)/P}}$$

Finally, the action is selected in a stochastic way, since a deterministic maximization would

lead to actions with very low precision and so a very high uncertainty. In fact, in real market context (those considered by dynamic pricing) we should try to be safe from uncertainty effects. For these reasons, we assign to each flat value a probability of being extracted equals to

$$\pi(x) = \frac{OF(x)}{\sum_k OF(k)}$$

Then, we select an action according to this probability and we will extract with higher probability the actions where precision is low. This behavior can be verified observing the red line in Fig. 4.27 where we use the same Unit Margin Market Model as in Fig. 4.24.

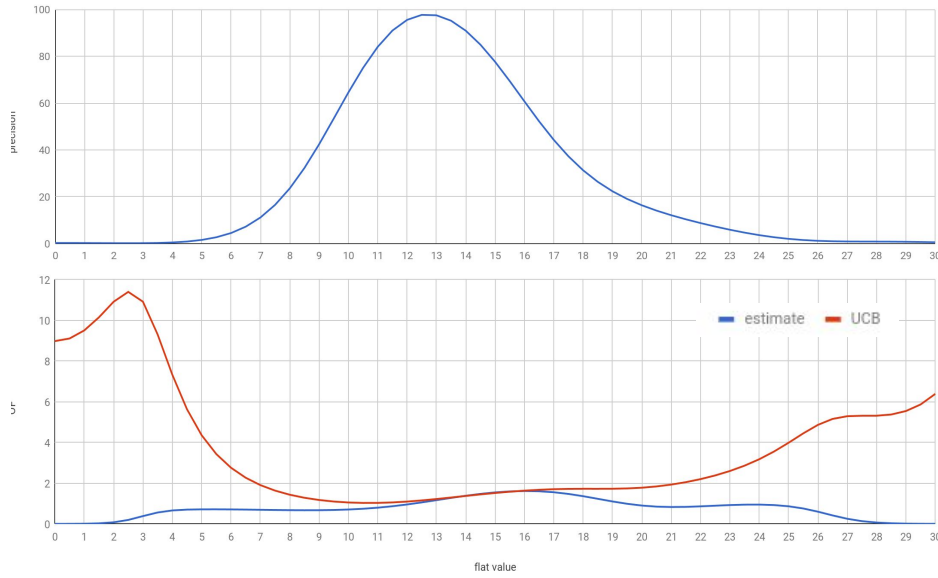


Figure 4.27: Objective function for the Upper Confidence Bound Policy based on a Unit Margin Market Model.

This method is very effective if we want to make a strong exploration of the environment, but this is not useful if there are some actions (with uncertainty effect) that could lead to bad business consequences, such as a loss in the revenues. Although this behavior could be attenuate by selecting only actions with a certain minimum value of precision, this policy could not really be used in dynamic pricing for its too high component of exploration in an uncertain environment. Nevertheless, it has been presented since it gives the idea for the next policy.

Margin-based Lower Confidence Bound Policy

Starting from the Unit Margin Market Model, we want to build a policy that penalizes actions with high uncertainty. We call this policy Lower Confidence Bound Policy, for resemblance with the Upper Confidence Bound Policy, but our technique does not have any theoretical relationship with confidence bounds.

In practice, the exploration-exploitation trade-off is solved by optimizing the weighted sum of the reward estimate and a function proportional to the precision. Even in this case we should select the parameter c that gives the weight of the precision with respect to the estimate. For this goal we divide estimate $m(x)$ and precision $\tau(x)$ by their maximum values:

$$M = \max_x m(x) \quad P = \max_x \tau(x)$$

Then, the objective function becomes

$$OF(x) = \frac{m(x)}{M} + c \left(\frac{\tau(x)}{P} \right)^\gamma$$

where γ is a parameter that attenuate the selection of exploration actions (usually it is $\gamma < 1$).

As before, an action is randomly selected by associating to each action a probability of being extracted:

$$\pi(x) = \frac{OF(x)}{\sum_k OF(k)}$$

In practice, this policy would prefer actions with a high estimate and a high precision. Actions with high estimate but low precision are selected as exploration actions, but thanks to the coefficient γ the selection of this action is done gradually, by starting exploring flat values near to known ones and moving forward to the unknowns.

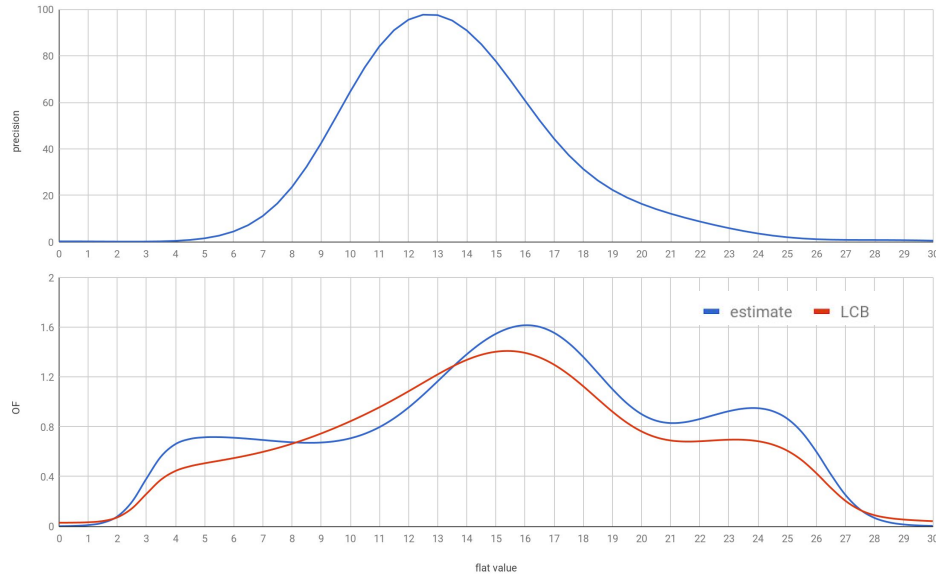


Figure 4.28: Objective function for the Lower Confidence Bound Policy based on a Unit Margin Market Model.

The shape of the objective function for this policy is the red line depicted in Fig. 4.28, where the Market Model is the same as before and we choose $\gamma = 0.5$ (i.e. we take the square root).

Finally, the Lower Confidence Bound Policy is useful when we want to mostly select actions for which we have an evidence that are good actions. Moreover, this policy would gradually explore other actions without jumping drastically in another region of the space, but moving forward by little steps. This is important because of the price perception of the users: we cannot change prices too drastically without a communication marketing.

The gradual exploration is motivated also by the fact that the market response has already been explored with market events coming from historical data. Then, we just need to follow temporal changes on the market and this can be done through non-drastic exploration actions.

4.5 Information propagation

As seen above, an important issue in using reinforcement learning to solve dynamic pricing problems is the lack of observations about the market response to different prices. As an example, in the insurance context, the partitioning of the flight booking into rigid groups (items) would limit the mass of information each one receives and consequently the statistical significance of the models. On the other hand, it could be useful to have a finer group division, in order to differentiate prices as much as possible while maintaining them homogeneous in terms of customer behavior.

Liquidprice tries to solve this kind of problems by an information propagation component called Wave. It helps to maintain a fine granularity of the models (large number of items) since it spreads information of what is happening in the market among different items, based on their similarity. In practice, events observed by a single item can be used to build the market model of similar items. How to compute the similarity between items is described below.

Another important feature of Wave is that its functionality is not context-dependent. In fact, the implementation is always the same, except for some configuration values that are proper of the item that we want to price. This non-custom implementation is useful for a cross-domain information propagation, that is the spread of information among quite different contexts (such as hotel and flight bookings).

4.5.1 Wave features space

In order to determine how similar items are, they are represented as points in a multidimensional space, called Wave features space. The dimensions of this space must be indicators which allow separating or uniting items in terms of how similar they are in customer behavior on purchasing that particular item.

In general, a system that stores data about purchases considers three types of features: product features, user features, and purchase features. The product features refer to the characteristics of a particular item that is sold, such as color and size for an object, additional services for a hotel room or origin and destination for a flight ticket. The user features are related to customer registry (if available) such as age, sex or city. Finally, the purchase features are represented by all the characteristic of a particular purchase of an item performed by a user. As an example, in the insurance context, we could have information about hour and date of the purchase, the time that is missing from the flight departure, the number of passengers for which the ticket is booked or if there are children in the reservation. Obviously, the price paid for an item is part of the purchase features too.

In order to use these features as dimensions for the Wave features space, for each item, we should compute some statistics starting from user and purchase features. For example, we could be interested in knowing if a product is bought by young or elder people, if the insurance

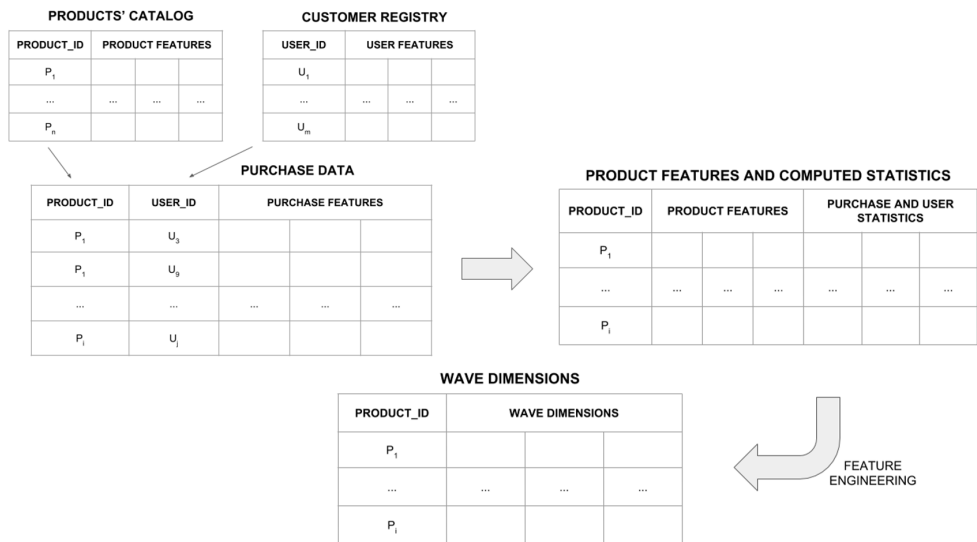


Figure 4.29: Transformation process for determining the dimensions of the Wave features space.

on a group of flight is bought in advance or as the flight departure is approaching, or if the purchase of an insurance on an item refers to one person or more.

This process of statistic computation ends up in a dataset containing for each item, the product features and the value of all the statistics computed starting from user and purchase features. These statistics represent the behavior of customers in purchasing different items.

Clearly, it's possible to compute a lot of statistics but not all of them are significant to explain these behaviors. In order to find the most important statistics that will be used as Wave dimensions, we build models that taking as input the products features and the computed statistics would predict the outcome of the purchase (if the item would be bought or not, or a probability of buying).

Performing a feature engineering work on these models, we could find a small set of product features and statistics that are the most important features in describing customers behavior. These features are then used as dimensions in the Wave features space.

Therefore, if we consider a multidimensional space with these dimensions, for each item, the values of these dimensions give the position of that item in the Wave space. Hence, two items are similar in terms of customer behavior on purchases if they are close with respect to a distance between points computed in the Wave space.

Finally, let us observe that items belonging to the same domain should have the same set of Wave dimensions, in order to compute comparable distances. For computing distances between items of different domains, we should choose a way to derive a distance from the available dimensions in the various domains.

4.5.2 Propagation dynamics

The dynamics of Wave for propagating the information are based on the distance between items. In practice, Wave propagates to "close" items market events relative to one item with a weight which is a function of the distance between the two items.

Let us consider an N -dimensional Wave features space and two items A and B. Each item has a position in the space defined by $a = \{a_i\}_i$ and $b = \{b_i\}_i$ for $i = 1, \dots, N$. Moreover, each item specifies a propagation threshold for each dimension ($\rho^{(A)} = \{\rho_i^{(A)}\}_i$ and $\rho^{(B)} = \{\rho_i^{(B)}\}_i$) and the maximum propagation range within which each item consider other items to be close ($r^{(A)}$ and $r^{(B)}$).

The propagation process has the following four steps.

Step 1. The item A receives a market event. The event is sent to all the other items in the space and each item has to decide if it should add that event to its market model.

Step 2. The item B computes its distance from A along each dimension:

$$\forall i = 1, \dots, n \quad d_i = \rho_i^{(B)} |b_i - a_i|$$

Note that since the item B has to decide if it should consider an event or not, we rescale the distance along each dimension by its propagation threshold (and not by the threshold of A).

Step 3. The item B computes the total distance from A as

$$d = \frac{1}{r^{(B)}} \left(\sum_i d_i^{p^{(B)}} \right)^{1/p^{(B)}}$$

where $p^{(B)}$ is a parameter defined by the item B that changes the type of distance that it computes. Potentially, each item could compute a different type of distance since each one could define a different parameter p .

Step 4. If $d > 1$, it means that the item B is too far from item A. Then there is not propagation from A to B and the item B should not add the event to its market model.

Otherwise, if $d \leq 1$, propagation could happen and the item B should add the event to its market model with a weight $w = f(d)$.

The function f should be such that $f(0) = 1$ and $f(1) = 0$ since we want the weight to be maximum for items very close to A and minimum for items that are far from A. A class of function with this behavior is

$$w = f(d) = (1 - d)^\gamma$$

where $\gamma > 0$ is a parameter defined by the market model and it controls the homogeneity among market models of different items. The higher γ we use, the more the models are nonhomogeneous since the weight of propagation is low. This means that events received by

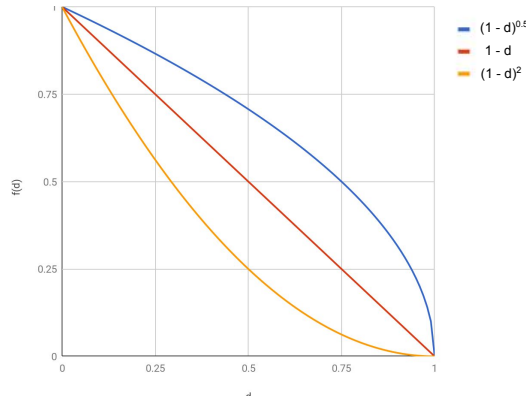


Figure 4.30: Different choices of γ in the function that transform distance into weight.

Wave have less importance than those received directly by the item. The effect of the parameter γ is depicted in Fig. 4.30: for $\gamma = 0.5$ the resultant weight is higher than the weight obtained with $\gamma = 2$. The use of $\gamma = 1$ is a common choice for having weight linearly proportional to the distance.

These four steps describe the general process of propagation. However, it's possible to characterize one or more dimensions as blocking dimension, which means that they could avoid propagation even if the total distance could have allowed it. This happens by adding a step to the previous process.

Step 2b. If for at least one of the blocking dimensions it holds $d_i^B > 1$, the item B is too far from item A along a blocking dimension. Then there is no propagation and the process stops. Otherwise, if for all blocking dimension it holds $d_i^B \leq 1$, the item B could compute the total distance and the process continues from Step 3.

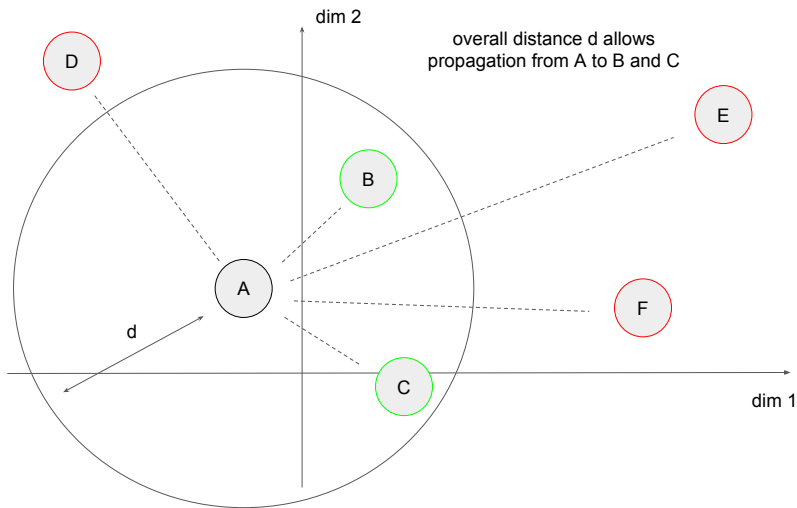


Figure 4.31: Wave dynamics: propagation allowed by overall distance.

Summarizing, the propagation among items happens if two conditions are satisfied at the same time:

1. The distance along each of the blocking dimension is less than its propagation threshold.
2. The overall aggregated distance is less than the maximum propagation range defined by the recipient item.

The two conditions are described in Fig. 4.31 and Fig. 4.32.

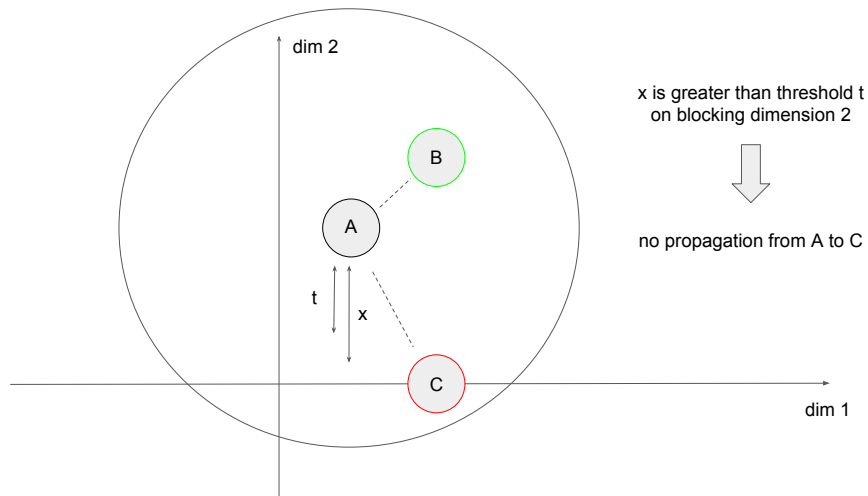


Figure 4.32: Wave dynamics: no-propagation due to blocking dimension.

The final result of the Wave propagation process is that we could have a market model for each point of the multidimensional space by interpolating the market models of close points. These models are more homogeneous as the maximum propagation range increases and the exponent of the function that transform the distance in weight decreases. Otherwise, if the range decreases and the exponent of the function increases, the market models are non-homogeneous.

5 Conclusions and future developments

At the end of this work, it's possible to conclude that reinforcement learning algorithms could be used for solving a dynamic pricing problem and defining a pricing strategy. In particular, a reinforcement learning approach allows to model in a more effective way the price-sensitivity of customers. This has been possible thanks to the dynamics of interaction between agent and environment, that are based on a trial-and-error action mechanism.

Starting from the idea of the most famous reinforcement learning algorithms, a new algorithm was developed. It is able to solve many problems that arise from the straightforward application of basic algorithms on dynamic pricing contexts. The new pricing algorithm, implemented in the software Liquidprice, consists of three main component: the market model, the pricing policy, and the information propagation engine.

The market model uses probability distributions for modeling price-sensitivity, in order to take into account the uncertainty of the customers' response over time. Moreover, each distribution (relative to one price) could influence other distributions, since information received for a single price could have weight also for close prices. This is a first trick to overcome the lack of information, due to the absence of available data.

After building the market model, the pricing policy plays the role of decision maker. In fact, it defines the rules that should be used for selecting the new price, based on the information stored in the market model.

Finally, the information propagation component represents the second trick for trying to solve the problem of lack of information. In fact, it helps to share information among different selling products, based on similarity of customers' behavior in purchasing those products. In this way, the number of purchase data observed by each product increases and the market model gets more statistical significance.

Chapter 5. Conclusions and future developments

The pricing algorithm proposed in this work gives the main idea for a dynamic pricing solution algorithm, without observing the context. As seen, there are many elements that could be improved or modified depending on the application context.

Future developments of this work will lead to build new types of market models using other kinds of value distribution or kernel function that are more specific for a particular application. Similarly, the set of pricing policies could be improved for taking into account different exploration-exploitation methods based not only on the multi-armed bandit problem, but also on the general Markov decision process.

References

- [1] G. R. Bitran and R. Caldentey, *An overview of pricing models for revenue management*, Manufacturing & Service Operations Management **5** (2003), 203–229.
- [2] R. Dearden, N. Friedman, and S. Russell, *Bayesian q -learning*, In AAAI/IAAI, AAAI Press, 1998, pp. 761–768.
- [3] M. H. DeGroot and M. J. Schervish, *Probability and statistics*, 4th ed., Person, 2012.
- [4] A.V. den Boer, *Dynamic pricing and learning*, Ph.D. thesis, 2013.
- [5] W. Elmaghraby and P. Keskinocak, *Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions*, Management Science **49** (2003), no. 10, 1287–1309.
- [6] K. J. Ferreira, B. H. A. Lee, and D. Simchi-Levi, *Analytics for an online retailer: Demand forecasting and price optimization*, Manufacturing & Service Operations Management **18** (2015), no. 1, 69–88.
- [7] S. Hormby, J. Morrison, P. Dave, M. Meyers, and T. Tenca, *Marriott international increases revenue by implementing a group pricing optimizer*, Interfaces **40** (2010), 47–57.
- [8] L. P. Kaelbling, M. L. Littman, and A. P. Moore, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research **4** (1996), 237–285.
- [9] K. Murphy, *Machine learning: a probabilistic perspective*, The MIT Press, 2012.
- [10] R. Rana and F. S. Oliveira, *Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning*, Omega **47** (2014), no. C, 116–126.
- [11] E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, The MIT Press, 2006.
- [12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, draft of 2nd ed., The MIT Press, 2017.
- [13] K. T. Talluri and G. J. van Ryzin, *The theory and practice of revenue management*, Springer Science, 2004.

References

- [14] W. T. L. Teacy, G. Chalkiadakis, A. Farinelli, A. Rogers, N. R. Jennings, S. McClean, and G. Parr, *Decentralized bayesian reinforcement learning for online agent collaboration*, In AAMAS, 2012, pp. 417–424.
- [15] G. J. Vulcano, G. J. van Ryzin, and R. Ratliff, *Estimating primary demand for substitutable products from sales transaction data.*, Operations Research **60** (2012), 313–334.
- [16] C. J. C. H. Watkins, *Learning from delayed rewards*, Ph.D. thesis, 1989.
- [17] J. Xu, P. Fader, and S. Veeraraghavan, *Designing and evaluating dynamic pricing policies for major league baseball tickets*, (2017).