Some contributions to decision-making problems

Frédéric Logé

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAX002

Thèse de doctorat

Some contributions to decision-making problems

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole polytechnique

École doctorale n°574 École Doctorale de Mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques appliquées

Thèse présentée et soutenue à Palaiseau, le 04/03/2021, par

FRÉDÉRIC LOGÉ

Composition du Jury :

Mme Mathilde MOUGEOT Professeure des universités, ENSIIE, affiliée à l'ENS Paris-Saclay, Centre Borelli	Présidente du jury
M. Aurélien GARIVIER Professeur, Ecole Normale Supérieure de Lyon, membre de l'UMPA et associé au LIP	Rapporteur
M. Sébastien GADAT Professeur des universités, Université Toulouse 1 Capitole, Toulouse School of Economics	Rapporteur
M. Karim LOUNICI Professeur, École polytechnique, CMAP	Examinateur
Mme Emmanuelle CLAEYS Maître de conférences, Université de Toulouse Paul Sabatier, I.R.I.T.	Examinateur
M. Erwan LE PENNEC Professeur, École polytechnique, CMAP	Directeur de thèse
M. Habiboulaye AMADOU-BOUBACAR Air Liquide R&D, Campus Innovation Paris	Encadrant de thèse
M. Eric MOULINES Professeur, École polytechnique, CMAP	Encadrant de thèse

Remerciements

"Tu vas apprécier en sortant d'une voie un verre d'eau... un sandwich. Et bon, ça c'est intéressant à l'époque à laquelle on vit d'avoir de petits besoins je crois. Tu as une prise de conscience au niveau des plaisirs simples en fait, qui suffisent très bien pour pouvoir vivre."
Patrick Edlinger, *La Vie au bout des doigts*

Mon coeur se pince à mesure que je rédige cette partie. Deux mois après ma soutenance. Je sens que la page se tourne. D'avance, pardon pour les oublis, que j'espère seront légers.

A mes encradants de thèse et mon jury

Du fond du coeur, je remercie mes encadrants de thèse Erwan, Eric et Habib. Merci à vous trois de m'avoir donné l'opportunité d'apprendre, de travailler libre, à mon rythme.

Eric, malgré le peu de fois où nous nous sommes croisés, à chaque fois tu m'as lancé sur des pistes toutes plus stimulantes intellectuellement les unes que les autres.

Habib, c'est -notamment- grâce à toi que j'ai pu faire cette thèse, et je t'en serai éternellement reconnaissant. Je suis très heureux d'avoir pu travailler avec toi et je me réjouis de continuer à le faire dans les années à venir. J'en profite pour remercier Jean André, Athanasios Kontopoulos, Martine Lanvierge-Mallet et tout autre membre de la R&D Air Liquide qui m'a permis d'être embauché pour cette mission de thèse.

Erwan, c'est -notamment- grâce à toi que j'ai pu finir cette thèse, et que je peux dire être fier de son contenu. J'ai eu grand plaisir à travailler avec toi sur ces sujets et je me réjouis de continuer notre collaboration dans ma nouvelle aventure. Un énorme merci pour ton soutien continu durant ces années, pour la liberté et la confiance que tu m'as accordé.

Mes remerciements viennent désormais à mes rapporteurs Aurélien Garivier et Sébastien Gadat, pour ce travail ingrat qu'ils ont acceptés de faire. Merci pour votre relecture, vos appréciations ainsi que vos nombreuses questions qui m'ont permis de prendre du recul par rapport à mes travaux. Au plaisir de vous recroiser en conférence et de collaborer sur quelques projets.

Merci à Mathilde Mougeot, examinatrice et présidente de mon jury, à Karim Lounici, examinateur, et Emmanuelle Claeys, examinatrice. Cela fut un plaisir de vous présenter mes travaux. Je vous remercie pour vos appréciations et vos questions, spécifiques et très intéressantes.

A ma famille

Merci à mes parents, Hubert Logé et Nelly Munerel, sans qui cette thèse n'aurait jamais été possible pour mille raisons. Vous êtes pour moi un modèle de travail acharné et pleins de

persévérance. Force à vous. Merci de m'avoir poussé tôt pour trouver ma voie.

Merci à ma soeur, Roxane Logé, une grande source d'inspiration et d'admiration pour moi. Tu es une personne merveilleuse et je te souhaite le meilleur dans la poursuite de tes rêves. Je suis derrière toi!

A mes professeurs, encadrants et étudiants

Un grand Merci à tous mes professeurs de l'IUT Paris Descartes, et notamment à Elizabeth Ottenwaelter, Olivier Bouaziz, Servane Gey, Florence Muri et Stéphanie Goujon. Merci à mes professeurs de l'ENSAI et de Rennes 2: Jocelyn Julienne, Laurent Di Carlo, Salima El Kolei, Fabien Navarro, Valentin Patilea, Adrien Saumard, Bernard Delyon, Myriam Vimond, Vincent Lefieux, Marian Hristache. Merci à tous pour vos cours de grande qualité ainsi que votre dédication à la pédagogie. Merci aux équipes administratives, sans qui rien ne tourne: Clarisse Pantin de la Guère et Anna Bela Ferreira au département STID de l'IuT Paris Descartes ; Corinne Barzic et Aurélie Duchesne notamment à l'ENSAI : mille mercis à vous !

Merci à Matthew Jackson de m'avoir accueilli chez Shareight et merci à Savithri Rajaraman de m'avoir introduit au web scraping, un outil dont j'ai usé et abusé depuis. Merci à Ingrid Sanchez pour ses bons conseils et sa bonne humeur contagieuse.

Merci à Charles Kervrann et Thierry Pécot de m'avoir accueilli à l'INRIA. Merci à Rémi Gribonval pour ses explications sur les techniques de reconstruction parcimonieuses en acoustiques.

Merci à Pierre-Marie Valton de m'avoir pris en stage chez Air Liquide et une reconnaissance éternelle à Moulay-Driss El Alaoui Faris, mon tuteur de stage à ce moment. Merci pour la liberté que tu m'as donné, pour tes conseils et ta disponibilité. Toujours impressionné par ta culture !

Je souhaite adresser un grand Merci à tous mes étudiants. Involontairement, ils m'ont beaucoup appris et continuent à m'apprendre beaucoup de choses, notamment sur moi. Ils me permettent de faire ce que j'aime probablement plus que tout: transmettre.

Merci notamment à Erwan, Elizabeth, Florence, Salima, Habib pour ces oppportunités d'enseignement.

Enfin, un grand Merci à Chirif Ousri, Wilfried Do Paco et Arthur Thomas, les meilleurs chargés de TD au monde.

Mention spéciale pour Arthur, c'est un plaisir de collaborer avec toi sur plein de sujets de recherche passionnant, ça commence à prendre une belle forme !

Aux collègues et aux copains[1]

Un grand Merci à tous mes collègues Air Liquide-ien qui ont rendu ces cinq années passionnantes de rebondissement: Armanda, pour ta gentillesse, Sameh, pour tes principes, Jean-Christophe, pour ce mélange subtil de cynisme et de gentillesse, Guillaume, pour ton coeur d'or, Benji, pour ton humour, Rodrigue, pour tes conversations passionnantes et ta rhétorique (courage pour la thèse !), François, pour tes blagues d'un certain goût, mais que j'apprécie tellement, Thomas, le Brice de Nice version Hipster et Stéphane, pour tes belles grimaces. Merci à vous tous, pour nos discussions, vos conseils et pour avoir été vous.

Mention spéciale pour l'équipe stagiaire & CDD: Vincent, Baptise, Matthieu, ça a été un plaisir de démarrer mon expérience pro à vos côtés, pour pleins de raisons. Une mention très spéciale pour Matthieu, super compagnon de randonnée, prodigueur de bons conseils - dans certains cas.

[1] Copain: personne avec une place importante dans mon coeur

Un grand Merci également à mes collègues du CMAP notamment Céline, Aude, Belhal, Florian F. et Florian B., sans qui ces bureaux de Palaiseau sont finalement bien tristes. Vous me manquez.

Merci à Nico, Fred, Pierre, Cédric et Vanessa, mes copains de l'IuT que je vois peu mais à qui je pense souvent. C'est dingue ça fait dix ans que je vous connais. Vous êtes tous des gens géniaux chacun à votre façon, et j'ai hâte de vous retrouver autour d'un verre. Mention spéciale à Nico pour tous ces petits moments très cool, de l'IuT, à l'Inserm, à l'Ensai, à aujourd'hui – t'es le meilleur !

Merci à Geneviève Robin pour nos moments de partage, tu es une personne magnifique. Je ne pourrais jamais te remercier assez de m'avoir introduit à l'escalade.

Merci à Rémi Besson, mon grand frère de thèse, pour nos moments de ping-pong comme de maths. Je suis tellement heureux de pouvoir travailler avec toi sur les fruits de ta thèse. Tu es impressionnant de travail, de connaissance et de sagesse ! Un grand Merci pour la confiance que tu m'accordes.

Merci à Wendy, Thibault, Floriane et Sandrine, mes compagnons de randonnée, bivouac et d'escalade. Un grand merci pour nos sorties de Nature dont je ne peux plus me passer ; hâte d'en passer encore une bonne centaine. Merci à Julie et Valentin pour une dernière à l'Ensai bien sympathique.

Mention spéciale pour Floriane : Merci de m'avoir fait découvrir l'escalade en extérieur, et Merci pour nos nombreuses sessions de bloc, hâte de continuer à grimper avec toi pour un bail ! Je te l'ai dis, mais là au moins c'est écrit: je suis ocnvaincu que tu peux percer jusqu'au 8ème niveau! et je vais root pour toi autant qu'il le faudra!

Un énorme Merci aux meilleurs colocs du monde: Pauline, Kevish, Juliette, Claire, Heythem.

Pauline, entre les soirées de babyfoot, l'appart dément, les courses de bon matin, le japonais en bas de l'appart, j'ai passé une année géniale, et c'est grâce à toi ! Hâte qu'on se retrouve !

Kevish, tu es juste trop cool et chill, hâte de te retrouver dans le Colorado qu'on se fasse un road trip !

Juliette, tu es passée bien vite, mais c'était des soirées bien cool ! Bon courage à toi dans tes nouvelles aventures !

Claire, tu es grave cool aussi, évidemment, mais en plus tu nous fais plein de patisseries ! Plus sérieusement, chapeau bas pour ta motivation et ta résilience, courage à toi !

Heythem, j'ai des soirées bien mémorables en tête quand je pense à toi, je suis tellement content que l'on ait pu tourner ensemble la page de la thèse ! Tu es si compréhensif et chill et sympa, je me languis de sortir faire la fête avec le Chef de la coloc !

Mary, sans toi j'en serais encore à la rédaction de mon manuscrit. Je pourrais jamais te remercier assez pour ton soutien et pour le bonheur que tu m'apportes. Tu es impressionnante de culture, d'intelligence et de détermination !

A l'attention des doctorants Les travaux présentés dans ce manuscrit sont issus principalement des travaux de ma dernière année. C'est OK si votre première année consiste en une pile de brouillons finalement "inutilisés". Bon courage pour votre aventure !

Contents

Notations

As a general note, random variables will be written with uppercase letters and there realizations will be written in lowercase, except for notations from Reinforcement Learning where we accept the notation use where variables and realizations are confounded. Caligraphed letters will be most often reserved for spaces. Most-used symbols are reported in table 1 below.

Symbol	**Meaning**
Supervised learning	
$Y, \mathcal{Y}$	target variable, space
$\hat{Y}, \hat{\mathcal{Y}}$	predicted target variable, space
$X, \mathcal{X}$	contextual information variable, space
$Z, \mathcal{Z}$	unobservable (yet) contextual information variable, space
$f, \mathcal{F}$	prediction function variable, space
$\hat{f}$	(data) fitted prediction function
$X \sim P$	X (a random variable), follows P (a distribution)
$\mathbf{L}, \mathbf{L}', \mathbf{L}''$	losses
Decision process	
$\mathcal{M}$	Markov Decision Process
$S, \mathcal{S}$	state variable, space
$A, \mathcal{A}$	action variable, space
$\mathbf{T}$	transition function
$R, \mathbf{R}$	reward variable, function
V	state value function
Q	state-action value function
π, Π	policy, policy spaces
θ, Θ	control variable, space
Miscellaneous	
$\triangleq$	definition
$\mathbf{K}$	kernel
$\mathbf{N}$	neighborhood function
card	cardinal
Gaussian(μ, σ)	Gaussian distribution with expectation μ, variance σ^2
Uniform(.)	Uniform distribution on either a closed space or a finite set

Table 1: Commony used notations. Indexing where omitted as they are multiple ways in which indexes are used for a given notation, and clearly specified in the text.

Introduction

General

In this thesis, we focus on diverse decision-making problems grounded on real-world applications, amongst which you will find: insulin management for type-I diabetics; an adaptive predictive-questionnaire for more seamless user experience; proper next-day demand forecasts for running a production unit.

Some of those applications require a single decision to be made, which we tackle with the *task-driven supervised learning*, presented in part I of the thesis. The key idea is that the loss function in the classic supervised setting is instanciated by the consequences of the forecast used to make the decision under the context uncertainty. This first major contribution fulfills the thesis project's initial aim of going from predictive to prescriptive view.

Now, several problems initially stated in the project consisted in either irregular episodic decision-making problems or quite cyclic decision-making problems. Both classes of problems were investigated under the frame of Markov Decision Processes, a well-established mathematical model, even more so now that (Deep) Reinforcement Learning is trendy. Our second major contribution, presented in part II, is to translate real-world problems into the vocabulary of Reinforcement Learning and Planning community, and build experiments, code and visualization to those applications. Essentially, we show what the technology can bring to the table to problems aside Atari games, albeit much less complicated, but of much more importance to the industry.

In the final part of this thesis, we explore special instances of Markov Decision Processes which do not require agent-enforced exploration to obtain good regret behavior. This part really stands out from the rest of the thesis, as it is not focused on real-world applications, but more on understanding why and when exploration is really necessary.

Each part of the thesis is preceded by its own introduction and each chapter is preceded by a short summary. For anyone interested in an overview of each parts, we provide summaries below. Following those, we give an overview of the applications studied in the thesis, followed by a note on the communications related to this project.

Converting predictions into action (part I)

The ability to predict different phenomenons has increased and gained a lot of attention outside the statisticians / machine learner community. There seems to be this belief that when one is able to predict this or that quantity, they will systematically have gained some advantage in taking better decision or strategy. In reality, when prediction is used as a stepping stone for decision-making, it is much more pertinent to incorporate the decision-making into the construction of the prediction algorithm. That way, we can actually evaluate the consequences of using the prediction as input for another program, which is the end goal. As we point out, the best approach, in terms of performance, is to directly model the decision-making without the forecast module. However, it is often the case that this would require to remake the decision-making module entirely, if it would even be possible: imagine a complicated routing optimization program, to which you add the stochasticity of customer demands.

In the first part of this manuscript, we use an extension of the classic supervised learning setting to one where a custom loss function, which is provided by the user, comes in the problem

to describe the consequences of making a prediction under certain circomstances. We call this new setting the *task-driven supervised learning* approach, which allows us to describe one-step decision-making problems i.e. we have some contextual elements available at a time, we make a prediction and later some decisions based on such prediction, until we observe finally some loss (or reward, equivalently).

This setting is developped in three different applications: medical wandering (chapter 2), forecast-based optimisation (chapter 4) and nominations on electricity grid (chapter 3).

Essentially two supervised learning approaches were adapted for this particular setting: decision trees, built with the custom loss, and function approximation methods in general, such as kernel approaches and neural networks.

Optimizing action sequences (part II)

Making one decision with important consequences at bay is a difficult task on its own, so imagine when you have to make decisions on a regular basis, as you would when piloting a production facility with a storage unit. You have to decide of a production plan and a storage plan to handle uncertainties related to your production, delivery system, and unexpected customer demands. In this setting, we rely on Markov Decision Processes and classic algorithms from Planning and Reinforcement Learning communities to attack our problem.

In the second part of this manuscript, we are interested in four different applications of Reinforcement Learning and Planning: two of those are for cyclic decision processes, see chapters 7 and 6, and the two others are for episodic decision processes, see chapters 8 and 9.

Exploration-free problems (part III)

The exploration-exploitation dilemma is a key topic in many decision-making problems where the system's behavior is only partially known and needs to be learnt in an efficient manner. This implies that one needs to tradeoff decisions for exploration i.e. selecting decision with the best **estimated** consequence and decisions for exploitation i.e. selecting decision to improve our knowledge as efficiently as possible. Note that all decisions can be considered on a spectrum of the (exploitation, exploration) grid, it is not one or the other. However, in specific processes we can show that enforcing exploration is not necessary to reach good regret behavior, either via simulation of plays or with regret analysis.

Details on applications

Let us expand on the applications of parts I and II.

Some single-step problems

Medical wandering (chapter 2) Rare disease are quite numerous and specialized centers exist, the difficulty lies in detecting issues in people's lives. We propose a medical knowledge-based approach joint with economic modeling of patient pathway to optimize overall balance between sending everyone to a specialized center and no one.

Nominations (chapter 3) On the electricity market, all suppliers nominate one day ahead the quantity of electricity they are ready to provide, and the next day an operator handles the settlement. Instead of nominating the actual quantity we believe we are going to produce (based on forecasts), we are integrating directly the loss function (especially the prices involved) to propose a pertinent nomination.

Forecast-based optimisation (chapter 4) It is quite common that in input of an optimisation program quantities yet unknown are required and need be forecast. We challenge the classic approach of building seperately forecast and optimisation problem and rather calibrate the forecast so as to optimize the final loss function.

Some multi-step cyclic problems

Forecast-based optimisation with storage unit (chapter 6) We re-investigate the problem of tailoring a forecasting function to its later use in the case where time-dependency is involved: the initial production facility is now equipped with a storage unit to allow for more flexibility. We examine how the initial methodology still works and go further by simply by-passing the forecast issue with an RL approach.

Diabetes (chapter 7) For bolus insulin injections, type I diabetes patients are often recommended to follow a standard bolus rule, handling current excess glucose and meal carbohydrate intake. This rule has not been justified other than from common sense and overall is very short-sided as it does not take into consideration the meal and activity plan of the person. Relying on a simulator and virtual patients, we investigate how to discover an optimal bolus advisor given a meal plan and later try to propose other meal plans which, optimized jointly with insulin management, bring lower glucose variability.

Some multi-step episodic problems

Protocol initial phase (chapter 8) We look at sequential milestones in the run-in of a device usage: the milestones are there to improve service quality close to the installation such that, in the long run, the device is used as much as possible by our user.

Intelligent Questionnaire (chapter 9) Acquiring data can be detrimental to user / patient experience, and so limiting the number of questions asked whilst maximizing the information level retrieved is an interesting goal for many areas. We develop a questionnaire, allowed a budget of questions, asking for information in order to maximize predictive power of an unknown target, as in the supervised learning setting.

Communications

The intelligent questionnaire work was partly presented at the UCAI'20 (User-Centered Artificial Intelligence) workshop, in September 2020. This workshop aimed at bringing together people from

the HCI (Human-Computer Interface) community and the Artificial Intelligence community.

The diabetes topic was presented at the Data Science for Health (DSHealth 2020) workshop which is part of the KDD conference. This workshop focused essentially on research topics which brought insights related to Health topics.

The medical wandering topic, done as a collaboration with R. Besson and S. Allassonnière, was accepted to the Journées de Statistiques (JdS 2020), postponed to 2021.

The topics related to nominations and forecast-based optimisation (industry) as well as protocol initial phase (health) where presented yet only internally at Air Liquide, with the objective of showing what could be potentially achieved for those topics, and spark off conversations and debate. Open contributions are in consideration.

Part I

One-step decision-making

Introduction

This part focuses on one-step decision-making problems where quantities are forecasted in order to take a decision, and some direct feedback is received and the process is repeated a large number of times. At the base of this chapter is the idea of combining supervised learning algorithms to loss functions tailored to the subsequent decision-making problem.

There seems to be a belief that "if we are able to predict a target Y, then we can take appropriate decisions". Appropriate, or even good decisions may be taken if target Y is perfectly known, however in practice it is rarely the case that we can predict a variable perfectly. Therefore, how we handle prediction errors i.e. how we define mathematically the cost of these errors plays an essential role in the predictor algorithm that is built.

Although this a well-known fact from the research perspective, not many flexible alternatives are proposed by the literature: typically, one might advise to use the mean absolute error instead of the mean squared error or look at a balanced accuracy instead of its classic version. We aim at letting the user define their own loss function, such that their predictor is a good predictor based on their definition. We regard this to be an important contribution also because the topic of the loss function is but rarely discussed in practice, even though it matters much more than the actual learning algorithm used.

Three applications were studied in this particular setting. In our first application, we predict whether somebody has a rare disease, and then choose whether or not we should send them to a specialized medical facility and therefore we look at the actual consequence (the incurred cost in that case) of our choice. In our second application, we consider the case where customer demand is forecasted from the prior day demand and production levels are adjusted to reach minimal costs. In the third application, we consider production nominations on the electricity market, taken into consideration the uncertainty over our own renewables production as well as future imbalance and day-ahead prices.

In all those cases, we look at the direct cost/reward of using a given prediction for a decision which has no long-term consequence on the system/environment it is interacting with.

Regarding the outline of this part: we start by recalling the classic supervised learning setting, talk about the importance of loss function choices and how to implement a supervised learning algorithms with a custom loss function. Then we present the applications, each in their chapter.

Chapter 1

From classic to task-driven supervised learning

Summary

In this first chapter, we recall the classic supervised learning setting (section 1.1), where one tries to build a predictor of $Y|X$ based on observed pairs $(x_i, y_i)_{i=1}^n$. Now, because this conditional distribution is but rarely deterministic, one must either specify what characteristic of this distribution is desired or choose a loss function, the two being sides of the same coin. In regression for example, amongst the limited choice of loss functions in available implementations, we often focus on the expectation (Mean-Squared Error), the median (Mean Absolute Error) or a quantile (Quantile error). This choice, aside from being intrinsically linked to the characteristic of $Y|X$ we will learn, is seldom challenged although it is what describes a predictor as *good*. We argue for the use of loss functions which are problem-specific and (therefore) user-defined.

Logically, in section 1.2 we introduce an extension of this classic supervised setting to a task-driven one, where the user specifies the loss function, which helps to quantify what is a good prediction in a given context. We also take a look at the case where a prediction is used as input of some black-box program and take the opportunity to introduce related works from the literature.

In section 1.3 we provide different practical approaches to build such predictors, which will be used throughout this part of the manuscript. The two first approaches are based on a proper calibration of decision trees and the last one relies on surface approximation, which we will perform using neural networks, although in small dimension it could be performed by other supervised learning algorithms.

1.1 Supervised learning and the role of the loss function

1.1.1 Classic supervised learning

Consider the couple (X, Y) where X, a random vector of dimension p, contains information to predict Y, a random variable. Often, Y will be reffered to as the target while X will be referred to as the features or context or predictors. The supervised learning task focuses on learning to predict Y based on X from a set of independent and identically distributed (i.i.d.) examples $(x_i, y_i)_{i=1}^n$ which come in batch. The term "supervised" refers to the fact that the target Y is observed, which stands in opposition to the "unsupervised" setting, where only features X are observed. See chapter 14 of Hastie et al. [2009] for more details on unsupervised learning.

Let us consider the dataset $\mathcal{D} = (x_i, y_i)_{i=1}^n$ i.i.d. realizations of (X, Y) under $\mathbb{P}(X, Y)$. We write $\mathcal{X}$ and $\mathcal{Y}$ as the domains of X and Y respectively and we will assume that $\mathcal{X}$ is a p-dimensional space and $\mathcal{Y}$ is a one-dimensional space. We shall write x_{ij} the j-th element of x_i. Consider $\hat{\mathcal{Y}}$ the target prediction space, which may match $\mathcal{Y}$ or something more complex related to this space.

Finally, consider a loss function $\mathbf{L} : \hat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}_+$ a function such that $\mathbf{L}(\hat{y}, y)$ which indicates how bad it is to give prediction $\hat{y} \in \mathcal{Y}$ when true value is $y \in \mathcal{Y}$. When $\mathcal{Y} = \mathbb{R}$, the regression case, it is common to take the squared error $\mathbf{L}(\hat{y}, y) = (\hat{y} - y)^2$. If $\mathcal{Y} = \{0, 1\}$, the binary classification case, it is common to predict $\mathbb{P}(Y = 1)$ and then the log-loss is more classic: $\mathbf{L}(\hat{p}, y) = y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})$.

A classic formulation of the supervised learning task is the following optimization problem:

$$f^* \triangleq \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(X,Y)}[\mathbf{L}(f(X), Y)] \tag{1.1.1}$$

where $\mathcal{F}$ is a family of prediction functions, which map $\mathcal{X}$ to $\hat{\mathcal{Y}}$.

Unless $\mathbb{P}(X, Y)$ is known, the previous equation is often replaced by its empirical version (without learning $\mathbb{P}(X, Y)$):

$$\hat{f}^* \triangleq \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} \mathbf{L}(f(x_i), y_i). \tag{1.1.2}$$

This classic setting is illustrated in figure 1.1. Please note that in this figure, and the ones following as well, we do not represent the potential links, causal or not, between X and Y, as this would just overcrowed the figure without adding much inside to it. Evidently, X and Y are not independent quantities, otherwise the study of $Y|X$ will be quite straightforward.

1.1.2 The loss function: regression case

A classic risk function is the squared prediction error $\mathbf{L}(\hat{y}, y) = ||\hat{y} - y||_2^2$, which, when averaged over observations is termed Mean Squared Error (MSE). It is also common to take its square root afterwards, leading to the Root Mean Squared Error (RMSE), which allows for comparisons on a proper scale for the target.

This metric makes some general sense: (a) the best prediction is to get the target exactly, (b) whether we diverge below or above, the risk is symmetric (without additional knowledge this is a reasonable assumption) and (c) the risk gets quite high quite rapidly to avoid big deviations. This choice is of high mathematical interest, as the squared function is convex, and in some parametric models we can even derive closed-form expressions. It is recognized as one of the most convenient and widely used metric for regression training.

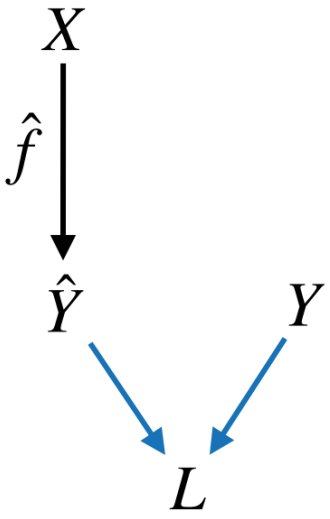

Figure 1.1: Classic supervised learning: based on X, a prediction $\hat{Y} = \hat{f}(X)$ is made. The prediction is then compared with Y, through loss function $\mathbf{L}$, giving L.

We have two points of contention:

1. it might be that the user, with the insight of how the predictions are later used, might want to define their own loss function, tailored to their problem

2. it might be that the user considers a "good" predictor as the median of $Y|X$, or its 75% quantile or something more complex as: if Y is positive, the focus is on the expectation of $Y|X$ and if Y is negative the focus is on 25% quantile for example (see illustration below).

Please note that in this work, we will focus on the first point of view, letting the user define a loss function tailored to their problem.

In both cases, the default choice of MSE is not enough and although those points of view are different they are intrinsically connected, for example: take

$$f^* = \arg\min_{f\in\mathcal{F}} \mathbb{E}_{(X,Y)}[\mathbf{L}(f(X),Y)] \triangleq \arg\min_{f\in\mathcal{F}} \mathbb{E}_{(X,Y)}[(f(X)-Y)^2] \tag{1.1.3}$$

$$= \arg\min_{f\in\mathcal{F}} \mathbb{E}_X\left[\mathbb{E}_{Y|X}[(f(X)-Y)^2|X]\right] \tag{1.1.4}$$

and so the optimal solution is to take, in every point $x \in \mathcal{X}$:

$$f^*(x) = \arg\min_{y\in\mathcal{Y}} \mathbb{E}_{Y|X}[(y-Y)^2|X=x] = \mathbb{E}_{Y|X}[Y|X=x] \tag{1.1.5}$$

which is the conditional expectation of Y given X.

Illustration: we simulated some i.i.d. pairs of (X,Y) under the following model:

$$\begin{cases} X \sim \text{Uniform}([-10,+10]) \\ Y|X \sim \text{Gaussian}\left(\sum_{j=0}^{p}\beta_j X^p, \sigma\right) \end{cases}. \tag{1.1.6}$$

Three different loss functions were considered, represented in figure 1.2 and defined as follows:

for any couple $(\hat{y}, y) \in (\hat{\mathcal{Y}}, \mathcal{Y})$,

$$\mathbf{L}(\hat{y}, y) = \begin{cases} (\hat{y} - y)^2 \ \textit{(Mean Squared Error (Average))} \\ \max\{.75(\hat{y} - y); .25(y - \hat{y})\} \ \textit{(Asymetric Absolute Error (.25 Quantile))} \\ \max\{y - \hat{y}; 0\}\mathbb{1}\{y > 0\} + \max\{\hat{y} - y; 0\}\mathbb{1}\{y \leq 0\} \ \textit{(Y-dependent Error)} \end{cases} \tag{1.1.7}$$

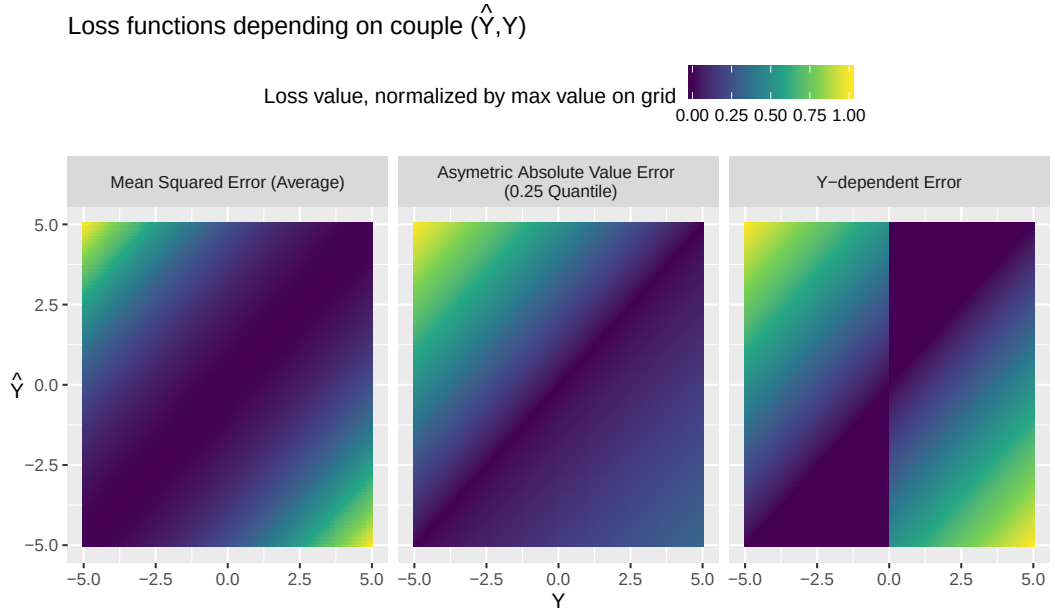

Figure 1.2: Maps of the three losses considered over the grid $(Y, \hat{Y})$.

The first two losses are quite common: the mean-squared error, which we mentionned earlier and a quantile loss, which, as the name suggests, estimates a conditional quantile. The last loss, taken arbitrarily, penalizes underestimating whenever Y is positive and penalizes overestimating otherwise.

In figure 1.3 we plot the data points simulated under model from equation 1.1.6 and added the predictors computed based on each loss function. The predictors were computed based on neighbour approximation of the optimization problem, specifically: for a given loss $\mathbf{L}$ and a given $x \in [-10, +10]$, we take

$$\hat{f}(x) \triangleq \arg\min_{\hat{y} \in \hat{\mathcal{Y}}} \sum_{l \in \texttt{N20}(x)} \mathbf{L}(\hat{y}, y_l) \tag{1.1.8}$$

where `N20` maps any $x \in [-10, +10]$ to the set of 20 observations from dataset $\mathcal{D} = (x_i, y_i)_{i=1}^n$ which are the closest by comparing the x component in the euclidian sense.

Note that each predictor captures something radically different on the distribution of $Y|X$, hence to the question *what is a good predictor ?* we answer: *whatever the end-user declares it is.*

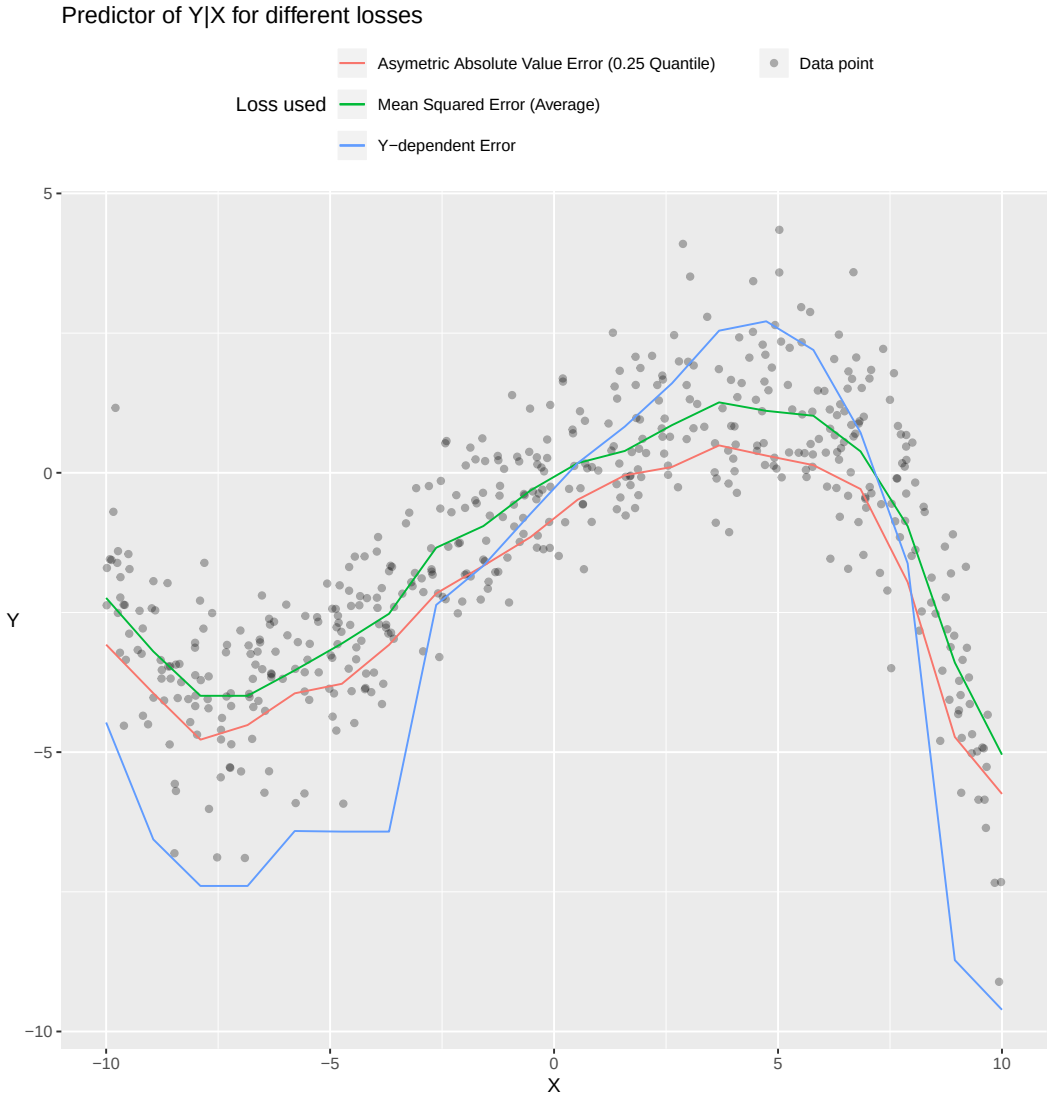

Figure 1.3: Three different predictors of $Y|X$, one per loss function, based on local approximations. For the third loss, the predictor tries to estimate the minimum value of $Y|X$ when $Y|X$ tends to take negative values and otherwise tends towards the maximum of $Y|X$.

1.1.3 The loss function: binary classification case

Note that in binary classification, one "only" needs to predict $\mathbb{P}(Y = 1|X = x)$, which is a single number and completely describes the distribution of the target. However, aside from small variants in loss functions, a classic issue comes from the class imbalance problem: depending on the proportion of positive ($Y = 1$) cases in your data, the model trained will reproduce this given percentage and it is often the case that a rare class is disregarded by the training procedure, unless some corrections is brought to it. Typically, one can set weights to observations, upsample the rare class data or downsample the dominant class data. In any case, it comes down to balancing out the classes, but one can also see it through a different perspective. Suppose the data is balanced, but the consequences of a false positive and a false negative are not equivalent in real-world application. In that case, similar strategies, and others, should be considered such that the output of the prediction function is aware of the consequences.

Illustration: we simulated i.i.d. sets (X, Y) under the following model:

$$\begin{cases} X \sim \text{Uniform}([-20, +20]^2) \\ Y|X \sim \text{Bernoulli}(\min\{p(X); 1\}) \\ \text{where } p(X) \triangleq .8\mathbb{1}\{d_2(X, (5, 5)) < 10\} + .7\mathbb{1}\{d_2(X, (-15, 0)) < 10\} + .15 \end{cases} \tag{1.1.9}$$

where

$$\begin{aligned} d_2 : \quad & \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_+ \\ & ((x_1, x_2), (o_1, o_2)) \mapsto \sqrt{(x_1 - o_1)^2 + (x_2 - o_2)^2}. \end{aligned} \tag{1.1.10}$$

The data generated is plotted in figure 1.4, with coordinates determined based on (X_1, X_2) and the color indicative of the class Y. Using a 20-nearest neighbour voting approach, we established the predictions presented in figure 1.5: for any $x \in [-20, +20]^2$, take:

$$\hat{f}(x) \triangleq \frac{1}{20} \sum_{l \in \mathtt{N20}(x)} y_l. \tag{1.1.11}$$

Now, from those predictions, one would like to classify a new data point in either class 0 or 1. The conversion from the probability estimated, $f(X)$, to this class is done by thresholding probability levels: the idea is to optimize in $\tau \in [0, 1]$

$$\mathbb{E}\left[(\text{Cost False Negative})\mathbb{1}\{Y = 1, f(X) \leq \tau\} + (\text{Cost False Positive})\mathbb{1}\{Y = 0, f(X) > \tau\}\right] \tag{1.1.12}$$

where (Cost False Negative) and (Cost False Positive) are positive constants specified by the user. One can show that, locally, the solution is to take

$$\hat{Y}(X) = \mathbb{1}\left\{\frac{f(X)}{(1 - f(X))} > \frac{\text{Cost False Positive}}{\text{Cost False Negative}}\right\}. \tag{1.1.13}$$

In figure 1.6 we represent the final decisions made for different cost ratios.

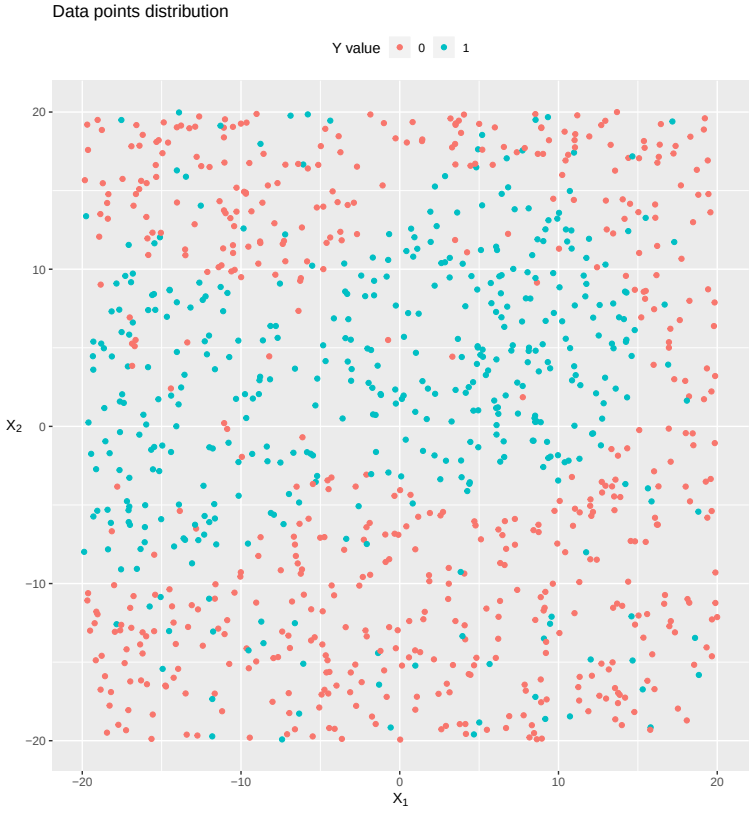

Figure 1.4: Data generated under model of equation 1.1.9.

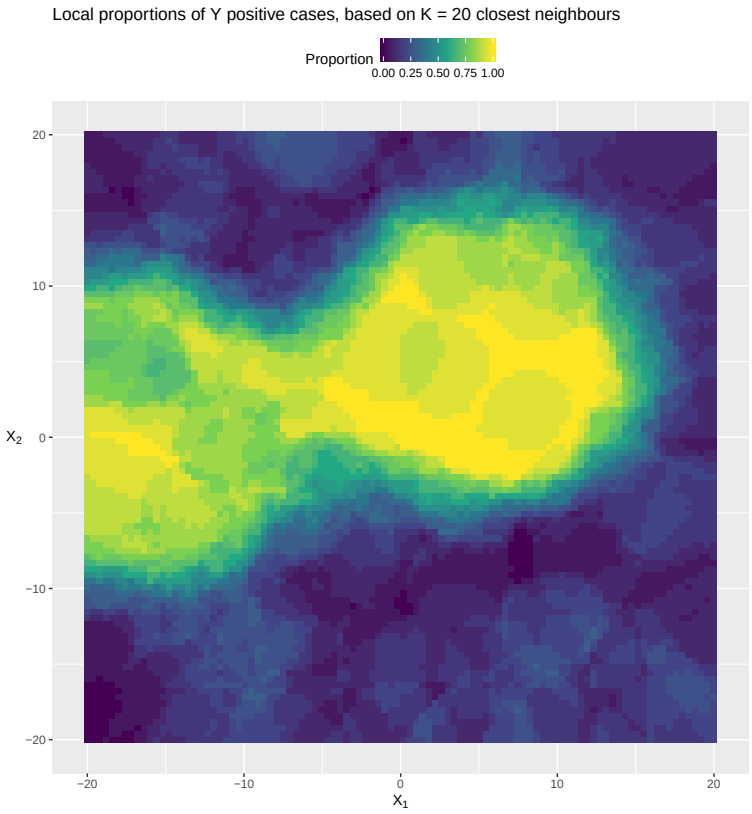

Figure 1.5: Predictions of $\mathbb{P}(Y = 1)$

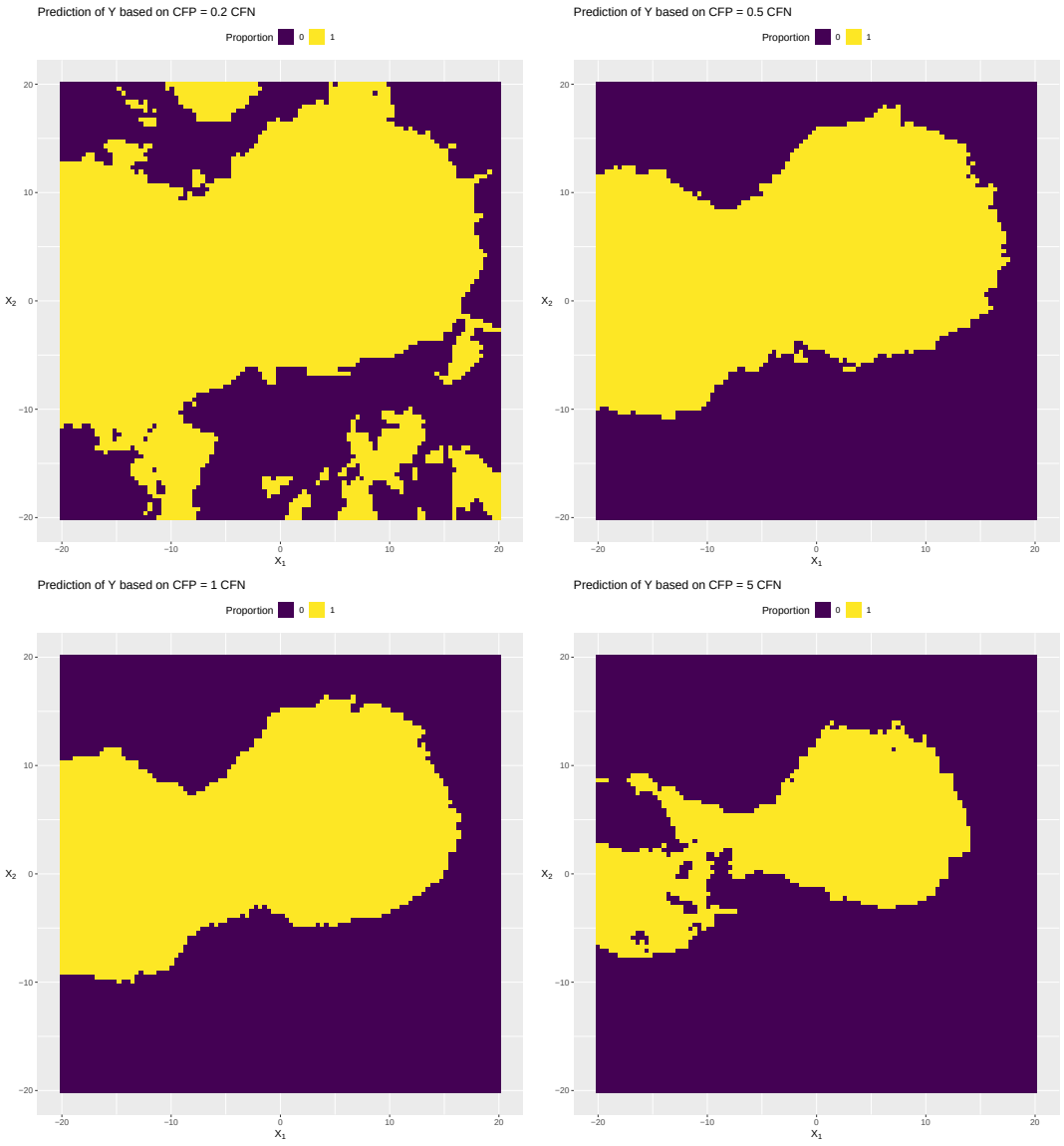

Figure 1.6: Prediction of class 1 for Y: as the cost of false positives increases (with respect to false negatives) we are progressively more conservative on our predictions of positive cases of Y.

1.2 Task-driven supervised learning

1.2.1 Task definition

Considering the supervised learning task expressed above, where we want to build a predictor of target Y based on X, we said that the risk function $\mathbf{L} : \hat{\mathcal{Y}} \times \mathcal{Y} \mapsto \mathbb{R}$ measures the quality of a prediction compared to the actual target. This risk supposedly embodies the consequence of actually taking a given prediction, not for the sake of prediction itself, but to actually use the prediction in a specific task, and the risk is then the incurred consequence faced with the reality (target Y).

Let $Z \in \mathcal{Z}$ denote information unavailable at the time of the prediction of Y but revealed jointly with it and let us write $\mathbf{L}'$ a loss/cost function indicating how bad predicting Y by $\hat{Y} = f(X)$ under context (X, Z),

$$\mathbf{L}' : \hat{\mathcal{Y}} \times \mathcal{Y} \times \mathcal{X} \times \mathcal{Z} \to \mathbb{R}_+. \tag{1.2.1}$$

This function will be referred to as our custom loss function, which we expect the user to define.

An example of custom loss definition is the one where weights are assigned to observations depending on observed values of (X, Y, Z) and the loss used is decomposed as:

$$\mathbf{L}'(\hat{Y}, Y, X, Z) = w(Y, X, Z)\mathbf{L}(\hat{Y}, Y) \tag{1.2.2}$$

where $w : \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}_+$ is the weight function and $\mathbf{L}$ is a classic error function.

Now, we define our task-driven supervised learning algorithm f^* as:

$$f^* \triangleq \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(X,Y,Z)} \left[\mathbf{L}'(f(X), Y, X, Z)\right] \tag{1.2.3}$$

where $\mathcal{F}$ is taken as family of prediction functions, mapping $\mathcal{X}$ to $\mathcal{Y}$. Note the differences with the classic supervised learning algorith framed in equation 1.1.1.

We assume that we have access to i.i.d. realizations of the triplet (X, Y, Z): $(x_i, y_i, z_i)_{i=1}^n$.

We assume that $\mathbf{L}'$ is known (e.g. user-designed) and we assume that for any subset I of $\{1, \ldots, n\}$ the following optimization problem is tractable:

$$\min_{\hat{y} \in \hat{\mathcal{Y}}} \sum_{i \in I} \mathbf{L}'(\hat{y}, y_i, x_i, z_i). \tag{1.2.4}$$

No knowledge is assumed of the joint distribution $\mathbb{P}(X, Y, Z)$.

This new setting is illustrated in figure 1.7.

1.2.2 Special case: forecast as input of BBO

Consider the case illustrated in figure 1.8 where the predictor output is used as input for an optimization program.

Associated to the optimization problem we consider loss

$$\mathbf{L}'' : \Theta \times \mathcal{Y} \times \mathcal{X} \times \mathcal{Z} \to \mathbb{R} \tag{1.2.5}$$

which outputs the cost incurred for true demand y in known context x and unknown context z taking control parameter $\theta \in \Theta$, Θ being the control space. Assuming the optimization problem is

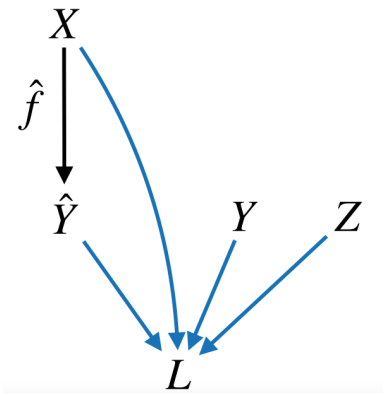

Figure 1.7: Task-driven supervised learning: based on X, a prediction $\hat{Y} = \hat{f}(X)$ is made. The prediction is then evaluated based on (X, Y, Z) through loss function $\mathbf{L}'$, giving L.

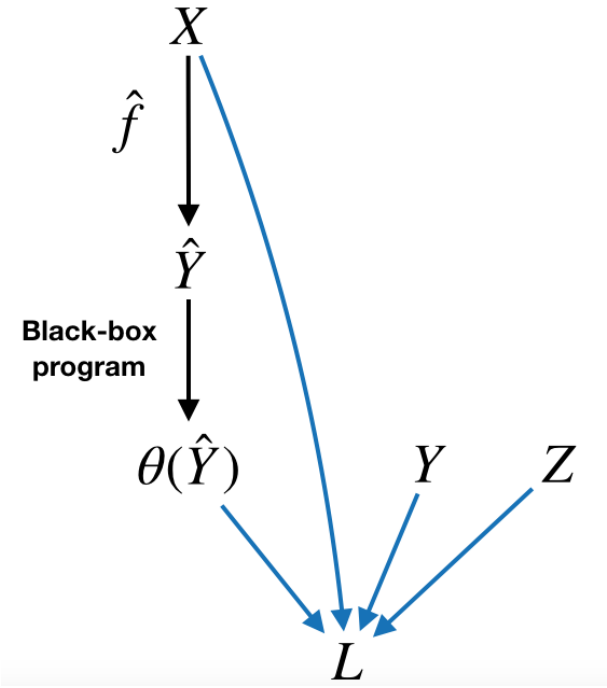

Figure 1.8: Task-driven supervised learning, forecasting for optimization. Based on X, a prediction $\hat{Y} = \hat{f}(X)$ is made, immediately used after by a Black-box program, outputting $\theta(\hat{Y})$. This final output is then evaluated based on (X, Y, Z) through loss function $\mathbf{L}''$, giving L.

tractable, we define

$$\theta^*(\hat{y}, x) \triangleq \arg\min_{\theta \in \Theta} \mathbb{E}_{Z|\{X=x, Y=\hat{y}\}}[\mathbf{L}''(\theta, \hat{y}, x, Z)]. \tag{1.2.6}$$

Taking $\mathbf{L}'(\hat{y}, y, x, z) \triangleq \mathbf{L}''(\theta^*(\hat{y}, x), y, x, z)$ in equation 1.2.3, we are optimizing the full chain, since:

$$\begin{aligned}
&\arg\min_{f \in \mathcal{F}} \ \mathbb{E}_{(X,Y,Z)} \left[\mathbf{L}'(f(X), Y, X, Z)\right] \\
&\quad \triangleq \arg, \min_{f \in \mathcal{F}} \ \mathbb{E}_{(X,Y,Z)} \left[\mathbf{L}'' \left(\arg\min_{\theta \in \Theta} \mathbb{E}_{Z|\{X,Y=f(X)\}}[\mathbf{L}''\left(\theta, f(X), X, Z\right)], Y, X, Z\right)\right].
\end{aligned}$$

Now, evidently, the best approach would be to remove the forecasting module and integrate information X directly in input of the optimisation program. The disadvantage is, evidently, that it requires changes to optimisation problem, which are generally complex enough as it is. Also, even if we improve overall costs by integrating the two problems, we could loose flexibility when new variables need to be considered and potentially much higher computational costs.

1.2.3 A baseline approach

A first approach to building a good predictor under a custom loss function $\mathbf{L}'$ would be to build a collection of K predictors $(f_k)_{i=1}^K$, focused on many different elements of $Y|X$ e.g. taking standard loss functions and built as independently as possible. From there, the problem is reduced either to finding the best predictor:

$$\arg\min_{k \in K} \mathbb{E}_{X,Y,Z}[\mathbf{L}'(f_k(X), X, Y, Z)]. \tag{1.2.7}$$

or the best convex combination for example:

$$\arg \min_{\substack{w \in \mathbb{R}_+ \\ ||w||_1 = 1}} \mathbb{E}_{X,Y,Z} \left[\mathbf{L}' \left(\sum_{k=1}^{K} w_k f_k(X), X, Y, Z\right)\right]. \tag{1.2.8}$$

Evidently this won't match the best possible predictor built using the loss function $\mathbf{L}'$ but it's a first simple step to take into consideration this custom loss.

1.2.4 Related works

In several research communities there seems to be a growing on this research topic[1].

In Donti et al. [2017], the authors demonstrate that neural networks can perform very well when trained as predictors of optimization problems. The results where obtained on three different operations research problems.

In Carriere and Kariniotakis [2019], the authors apply an end-to-end approach for the optimisation of their system which is composed of (a) the forecast of uncertain quantities and (b) a trading problem. This approach requires to dig deep within the problem, for instance they rely on Particle Swarm Optimization to solve the global problem, which is a much more complex thing to do than what we propose here. Also, it seems that the forecast in itself cannot be trained seperately from the final trading problem, which in our case could be.

[1] As of now, it seems no name has yet been coined on it: on our part, we will most often speak of task-driven supervised learning.

In Huang et al. [2019], the authors use a reinforcement learning framework to calibrate the prediction algorithm as close as possible to evaluation metrics chosen. They manage to direct the learning towards non-decomposable metrics such as Average Precision-Recall in the classification setting. The overall procedure seems quite heavy, yet very adaptive.

In Bertsimas and Kallus [2020], the authors worked on a similar problem as defined in equation 1.2.3:

$$\arg\min_{\hat{y}\in\hat{\mathcal{Y}}} \mathbb{E}_Y \left[\mathbf{L}(\hat{y}, Y)|X = x\right]. \tag{1.2.9}$$

They propose to adapt classic ML algorithms, notably kNN, CART and random forests to this objective. This formulation is certainly the closest to ours, although we add the consideration of unobserved contextual information Z in the loss function.

1.3 Methods

In this section we propose three approaches to approximate f^* as defined in equation 1.2.3: the first consists in tweaking the predictions from decision trees terminal nodes, the second consists in rebuilding decision trees based on the custom loss, and the third consists in estimating the expected loss for a given predictor, and then finding the appropriate predictor, which is a standard practice in Reinforcement Learning (approximating Q-values).

1.3.1 Tweaking DTs terminal nodes prediction

Consider a decision tree built to predict Y given X using any approach available. This tree partitions the space $\mathcal{X}$ and assigns to any element x of $\mathcal{X}$ a terminal node final_node(x).

For all terminal leaves of the tree, we can recalibrate the associated prediction based on the training data within the leaf as

$$y^*\left((x_i, y_i, z_i)_{i\in\text{leaf}}\right) \triangleq \arg\min_{\hat{y}\in\hat{\mathcal{Y}}} \sum_{i\in\text{leaf}} \mathbf{L}'(\hat{y}, y_i, x_i, z_i). \tag{1.3.1}$$

From there, for any $x \in \mathcal{X}$, one has the following predictor:

$$f^*(x) \triangleq y^*\left((x_i, y_i, z_i)_{i\in\text{final\_node}(x)}\right) \tag{1.3.2}$$

Note that, aside from changing the loss function and adding contextual information (x, z), this is classic approach. Note that if we can build decision trees for a specified loss, we can also apply aggregation methods, typically bagging (random forests).

1.3.2 Building tweaked decision trees

Building a decision tree with a specific loss function can be done following the steps in algorithm 1 as is done classically.

Algorithm 1: Build Decision Tree with custom loss function

input initial data $(x_i, y_i, z_i)_{i=1}^n$
input loss function $\mathbf{L}'$
input required tree depth max_depth
input splitting parameters (e.g. number of data points per split)
for $d = 0, \ldots, \text{max\_depth} - 1$ **do**
    Get list of all leaves at depth d
    Find best split for each leaves, following loss function $\mathbf{L}'$
    Save all leaves for the following depth
end
Compute prediction for each terminal leaves
return splits and final predictions

For a given leaf of the tree i.e. a subset of observations, the prediction based on the training data within the leaf is defined as

$$y^* \left((x_i, y_i, z_i)_{i \in \text{leaf}} \right) \triangleq \arg\min_{\hat{y} \in \hat{\mathcal{Y}}} \sum_{i \in \text{leaf}} \mathbf{L}'(\hat{y}, y_i, x_i, z_i) \tag{1.3.3}$$

and the according minimal loss is defined as:

$$\text{sum\_losses}^* \left((x_i, y_i, z_i)_{i \in \text{leaf}} \right) \triangleq \min_{\hat{y} \in \hat{\mathcal{Y}}} \sum_{i \in \text{leaf}} \mathbf{L}'(\hat{y}, y_i, x_i, z_i). \tag{1.3.4}$$

Now, the best splitting criterion for a given leaf can be defined as

$$\begin{aligned} (j^*, \tau^*) \left((x_i, y_i, z_i)_{i \in \text{leaf}} \right) \triangleq \arg\min_{(j,\tau)} \quad & \text{sum\_losses}^* \left((x_l, y_l, z_l)_{\{l \in \text{leaf} \,:\, x_{lj} < \tau\}} \right) \\ & + \text{sum\_losses}^* \left((x_l, y_l, z_l)_{\{l \in \text{leaf} \,:\, x_{lj} \geq \tau\}} \right) \end{aligned} \tag{1.3.5}$$

Of course, adjustments can be made to balance data size between child splits.

The predictor built is the same as the one previously expressed in equation 1.3.2, except of course the terminal nodes are different as the tree is built differently.

As was mentionned in the previous case, if we erase (X, Z) from the loss metric, and if we consider only classic loss metric, as quite often only those are available in implemented packages. Also, if we can build decision trees for a specified loss, we can also apply aggregation methods, typically bagging (random forests).

1.3.3 Learning expected loss

Finding f^* is equivalent to find

$$\forall x \in \mathcal{X} \quad f^*(x) = \arg\min_{\hat{Y} \in \mathcal{Y}} \mathbb{E}_{(Y,Z)|X=x}[\mathbf{L}'(\hat{Y}, Y, X, Z)]. \tag{1.3.6}$$

To do so, we could start by approximating

$$\forall (x, \hat{y}) \in \mathcal{X} \times \hat{\mathcal{Y}} \quad g(x, \hat{y}) \triangleq \mathbb{E}_{(Y,Z)|X=x}[\mathbf{L}'(\hat{y}, Y, X, Z)] \tag{1.3.7}$$

by a supervised learning algorithm and then solve

$$\forall x \in \mathcal{X} \quad \arg\min_{\hat{y}\in\hat{\mathcal{Y}}} g(x,\hat{y}) =: f^*(x). \tag{1.3.8}$$

This is presented in algorithm 2.

Algorithm 2: Learn expected loss

input initial data $(x_i, y_i, z_i)_{i=1}^n$
input loss function $\mathbf{L}'$
input $J \geq 1$ number of simulated forecasts per observation
simulate augmented data $(x_i, \hat{y}_{ij}, y_i, z_i)_{\substack{i=1,\dots n \\ j=1,\dots J}}$
where $\hat{y}_{ij} \sim \text{Uniform}[\min_{i\in\{1,\dots,n\}} y_i, \max_{i\in\{1,\dots,n\}} y_i]$
compute for each observation the loss $\mathbf{L}'(\hat{y}_{ij}, y_i, x_i, z_i)$
train supervised learning model for target: loss vector, features: $(x_i, \hat{y}_{ij})$

The choice of approximator can be left to the user, in this thesis we will use kernel methods for low-dimension problems and neural networks. Kernel methods are reasonable choices as long as sufficient data is available with respect to the dimension. Most often, it will require to have $\dim(\mathcal{Y}) + \dim(\mathcal{X})$ to be not larger than 3.

Note that this approach of learning the expected loss and then optimizing over it is akin to what is done in reinforcement learning/planning with state-action value function approximations in the case where immediate rewards matter. The prediction $\hat{Y}$ corresponds then to the action taken in state X. Those approaches will be presented in more detail in the second part of this manuscript, as they will allow us to deal with multi-step decision-making i.e. settings where there are not-immediate repercussions to our actions.

Chapter 2

Rare-disease patients Medical Wandering †

Summary

Medical wandering is a frequent phenomenon, whether people have unexplained symptoms, or actually suffer from a disease which requires a specialist to set a diagnosis. In this work, we set a framework were an appropriate balance may be found between the wandering costs and the use of specialized services. This framework is based on a precise degree of knowledge of which symptoms tend to declare when a patients suffers from a given disease and most importantly, when. We create an algorithm, which, following the digitalized and regularly updated medical file of symptoms of a patient, generates an alarm when the suspected probability of a rare disease is raised, such that the patient may be treated appropriately and as early as possible.

2.1 Context

The recent Erradiag[1] report from the Rare Disease Alliance showed that in France a patient suffering from a rare disease[2] will have to wait an average of 2 years between the appearance of the first symptoms and the diagnosis of his pathology. For more than a quarter of them, this duration will be more than 5 years. This lapse of time, known as diagnostic wandering, is a scourge for the healthcare system due to the additional economic (multiplication of unnecessary medical examinations and treatments) and human costs generated (aggravation of the pathology due to inadequate care).

Patients suffering from a rare disease are particularly affected by erroneous diagnosis due to the multifactorial nature of these diseases: a rare disease often affects several different organs and diagnosis therefore requires a multidisciplinary approach. Moreover, a practitioner not practicing in an expert center will probably never observe most rare diseases during his career. However, although rare diseases are by definition infrequent, there are many patients, with an estimated 3 million people affected in France and between 263 and 446 million worldwide [Nguengang and al, 2019]. This is due to the very large number of rare diseases: no less than 9000 are listed in OrphaNet[3], the portal dedicated to rare diseases.

France is a pioneer in the fight against rare diseases, launching in 2005 the first National Plan for Rare Diseases with the structuring of Centres of Reference for Rare Diseases. The aim of these multidisciplinary centers is to provide better care for patients suffering from rare diseases. The Erradiag report reminds us that, despite the progress made following the creation of these centers, a still too large number of patients are still referred to a hospital structure very late. A quarter of patients thus wait more than 4 years after the first symptoms appear before the search for a diagnosis is finally initiated. A better orientation of patients in the healthcare system is therefore necessary.

At the same time, medical data, in the form of an expert database such as Orphanet or patient path data (consultations/reimbursements) with the advent of the electronic medical record, have been structured and are now more accessible for research, making it possible to achieve the objective of improving patient paths.

There are a number of expert databases for rare diseases, in particular OrphaNet. This database references more than 9000 rare diseases, and associates them with their characteristic phenotypic signs. This means that for each rare disease we know the symptoms generally associated and we know the probability of presenting the symptom knowing the rare disease concerned. This database is interoperable with HPO [Köhler and al, 2016] which provides a unified vocabulary of phenotypic signs and associated ontology. Access to patient pathway data is more difficult since it is personal data. The Rare Disease Plan foresees the deployment of the National Bank of Rare Diseases, interoperable with HPO and Orphanet, to collect data on patient pathways within rare disease centers. However, this data does not provide, or only partially, the medical path before entering an expert center. Patient pathway data (Electronic Health Record in the literature or EHR) are available online but the data are not labeled with an Orpha identifier for rare disease patients. Some works such as [Colbaugh et al., 2018, Tremblay et al., 2018] circumvent this difficulty by annotating the EHR database with expert information (for example if a drug has been taken it is

[1] *L'errance de diagnostic : Erradiag résultats de l'enquête sur l'errance de diagnostic. 2016.* 844 patients where surveyed.

[2] Rare disease : disease whose prevalence is below the 1/2000 threshold (European standard).

[3] https://www.orpha.net/consor/cgi-bin/index.php. Accessed on [23/02/2020].

probably a rare disease) but such an approach can only be limited to certain rare diseases.

The link between expert databases such as OrphaNet and patient pathway databases is not easy to operate. Indeed, while OrphaNet associates symptoms (HPO identifiers) with rare diseases (Orpha identifiers), the EHRs reflect the health pathway (drug reimbursement, general practitioner/specialist consultation). A first challenge is to associate an HPO symptom to an EHR health event. [Zhang et al., 2019] proposes an NLP approach to annotate the free text of EHRs with HPO identifiers. We hypothesize here that such a link is possible and that it is thus possible to combine expert data with clinical data, which is essential in the case of rare diseases where clinical data alone are not sufficient but allow the introduction of a notion of temporality absent from Orphanet.

In the section 2.2 we present our mathematical modelling of the decision making problem: to send or not to send a patient to a specialized center. In the section 2.3 we present the application of our approach on simulated patient pathways, for which the R code is freely available. We then follow with a discussion of the results and a general conclusion.

2.2 Modeling

A patient suffering from a rare disease in France has to wait an average of two years before being diagnosed. This medical wandering is highly detrimental both for the health system and for patients whose pathology may worsen. There is an efficient network of specialized centers but patients are only referred to these structures too late. We are considering a probabilistic modelling of the patient pathway in order to create a simulator that will allow us to create an alert system that detects wandering patients and refers them to a specialized center while considering the potential additional costs associated with these decisions.

2.2.1 Patient pathway

We consider an individual to be at risk of contracting a syndrome[4] on a given day and assume that a most one syndrome declares itself during the time period. The patient only suspects the presence of this syndrome through the onset of symptoms. We note then S_i, $i \in \mathbb{N}$, the set of symptoms observed on the i-th day following the onset of the disease and $H_t = \{S_i; 1 \leq i < t\}$ the history of these observations until day $t-1$. An example of a patient pathway is shown in figure 2.3, top graph.

2.2.2 Symptom types

We assume that the symptoms are divided into three groups:

- Latent symptoms, which are not observable without proper medical examination e.g. cardiac or neurological malformations.
- Permanently visible symptoms e.g. external malformation.
- Recurrent and transient symptoms. These are symptoms that may recur and then disappear and reappear e.g. migraines, ear infections.

[4]Syndrome is considered as a synonym of disease

For this study we simulated a graph linking syndromes and symptoms, as well as the probability of symptom occurrence over time, as shown in figure 2.2, top graph. We have assumed that the times of occurrence follow a Gaussian law, truncated on the left by 0. For chronic type symptoms, we assumed that the delays between presence/absence of symptoms follow an exponential law.

2.2.3 Decision-making

Based on the history of the individual, and following the onset of at least one symptom, we will check whether it is relevant for the patient to go to a center specializing in rare diseases.

Specifically, we assume that we are able to establish a predictor $\hat{f}$ of the probability that an individual is affected by a rare disease based on his history H_t. More precisely, we will rely on two fundamental indicators: for each symptom, has it already been observed or not and if it has, how long ago.

From the onset of the first symptom, we check each day if, yes or no, this probability exceeds a threshold $\tau, \tau \in [0,1]$, previously set. If the threshold is exceeded, the patient is sent to the specialized center and his syndrome, rare or not, will be discovered. There will be a cost to the specialist physicians and the tests performed. If the threshold is not exceeded, the patient will not be sent to the specialized center and will be sent at the end of the period for a regular checkup. We note π_τ this procedure which to a history H_t associates the binary decision $\mathbb{1}\{\hat{f}(H_t) > \tau\}$, and illustrate it in figure 2.1.

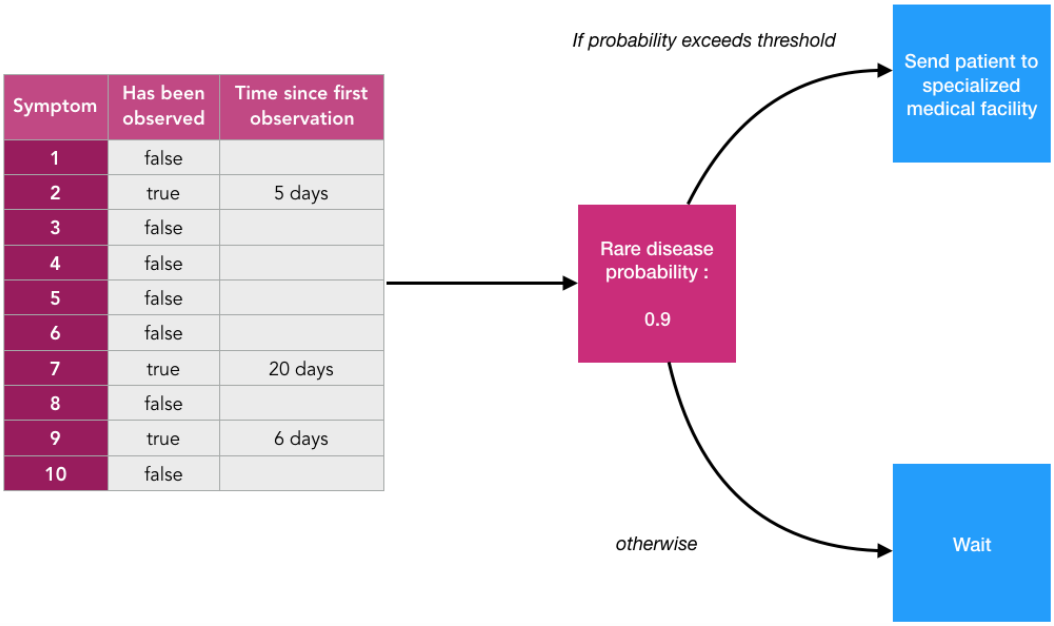

Figure 2.1: Schematic of the process

Consider the events E = "The patient has a rare disease" and A = "We referred him to a rare

Name	Label	Value
costSpecializedMD	cost of rare-disease-specialized MD	200 u
costNonSpecializedMD	cost of non-rare-disease-specialized MD	75 u
costWanderingPerDay	daily cost of wandering	0.08 u
meanTimeWandering	average wandering time	1460 days
aveNbMDConsulted	average number of MDs consulted	5

Table 2.1: Problem constants chosen (arbitrarily) for our simulations. The most important is that a specialized MD is considered to cost much more than a non-specialized MD.

disease center". We can then define a cost for a patient pathway based on these two events:

$$\textbf{cost}(\text{timeWandering}) = \begin{cases} costWanderingPerDay \cdot \text{timeWandering} & \\ \quad + costSpecializedMD & \text{if } E \cap A \\ costWanderingPerDay \cdot meanTimeWandering & \\ \quad + costNonSpecializedMD \cdot aveNbMDConsulted & \text{if } E \cap \bar{A} \\ costWanderingPerDay \cdot \text{timeWandering} & \\ \quad + costSpecializedMD & \text{if } \bar{E} \cap A \\ costWanderingPerDay \cdot \text{timeWandering} & \\ \quad + costNonSpecializedMD & \text{if } \bar{E} \cap \bar{A} \end{cases} \tag{2.2.1}$$

where timeWandering is the time between the observation of the first symptom and the decision making (if decision making, otherwise last day of the simulation). Detailed costs, listed in table 2.1 can be collected from medical experts.

Please note that by defining Y as $\mathbb{1}\{\text{"The patient has a rare disease"}\}$ and considering the wandering time as an element of X, aside from the transformations of H_t already mentionned (check if symptom has been observed, if it is how long ago), one may rewrite the previous cost function with the formalism of chapter 1.2.

Our objective is to determine, for a rare disease predictor $\hat{f}$ the appropriate τ threshold to optimize the cost defined in equation 2.2.1. Formally, we seek to find

$$\tau^* = \arg \max_{\tau \in [0,1]} \mathbb{E}_{\text{syndrome} \sim \nu} [\textbf{cost} \mid \text{syndrome}] \tag{2.2.2}$$

where ν is the probability distribution of the syndromes in the population.

2.3 Results

Full details on the results and simulations presented here can be obtained at `github.com/FredericLoge/patientPathway`. Seeds have been carefully established in the code to ensure reproducibility.

To facilitate the study, we simulated a symptom-syndrome graph as well as the probabilities of occurrence over time of the different symptoms as represented in figure 2.2. Four syndromes were considered, including one RAS (Nothing to Report) and one belonging to the category of rare diseases, the syndrome #1. A total of 10 symptoms were considered.

We generated for each syndrome 100 patient pathways over a four-year period, at daily time steps. These data were used to calibrate a random forest predicting the likelihood that the patient

being monitored has a rare disease based on which symptoms were observed, and how long it has been since the first symptom appeared. Once the predictor is trained, we can estimate the objective function expressed in the equation 2.2.2 on a fairly fine τ grid, as shown in Figure 2.3. On the graph on the left, the last graph shows the predictor is action, used as an alarm system. On the right graph, we can see the average cost as a function of the chosen threshold. As expected, there is a balance to find between sending everyone in specialized center (τ =0, far left of the graph), and sending no one (τ =1, far right of the graph).

2.4 Conclusion

In France, a patient suffering from a rare disease has to wait an average of two years before being diagnosed. This medical wandering is associated with high economic and human costs, even though expert centers for rare diseases have been set up in the country and give satisfaction when they are called upon.

We have therefore proposed here a procedure for the creation of an alert system to detect in databases of patient pathways patients suffering from rare diseases and needing to be referred to expert centers. We take into account in these decisions the different costs generated by the various possible actions. This alert system is trained by generating data from a patient pathway simulator that we build from expert data and whose parameters will have to be calibrated with clinical data. The first results, obtained on simulated data, appear promising. However, much work remains to be done on the following points.

Calibrating the Symptom-Syndrome Model This model must eventually be calibrated using a mixture of expert data and patient pathways. This data can be combined at the cost of significant work. The aim is to build a module associating an HPO identifier to each health event of an EHR. An expert annotation should also make it possible to better model the appearance of symptoms over time: information on the typical age of appearance of a symptom, the typical durations, the nature of the symptom.

Improve and validate modeling In our patient pathway model, the physician was present only after the decision to go to a specialized center was made or the simulation time was over. In reality, the occurrence of a symptom will likely prompt the person to consult a medical professional. It is imperative that these agents be integrated into the patient pathway. Indicator-based validation approaches such as the one developed in [Prodel, 2017] will be used.

The control problem We have considered here a particular strategy for redirection to expert centers that performs a binary choice if a certain threshold of probability is exceeded that the patient presents a rare disease. A future direction is to formulate the problem through a Markov decision process where the strategy can be learned by reinforcement learning algorithms based on simulator data and no longer necessarily having such a specific form.

Acknowledgements : This work has benefited from State aid managed by the French National Research Agency under the Future Investments programme bearing the reference ANR-19-P3IA-0001.

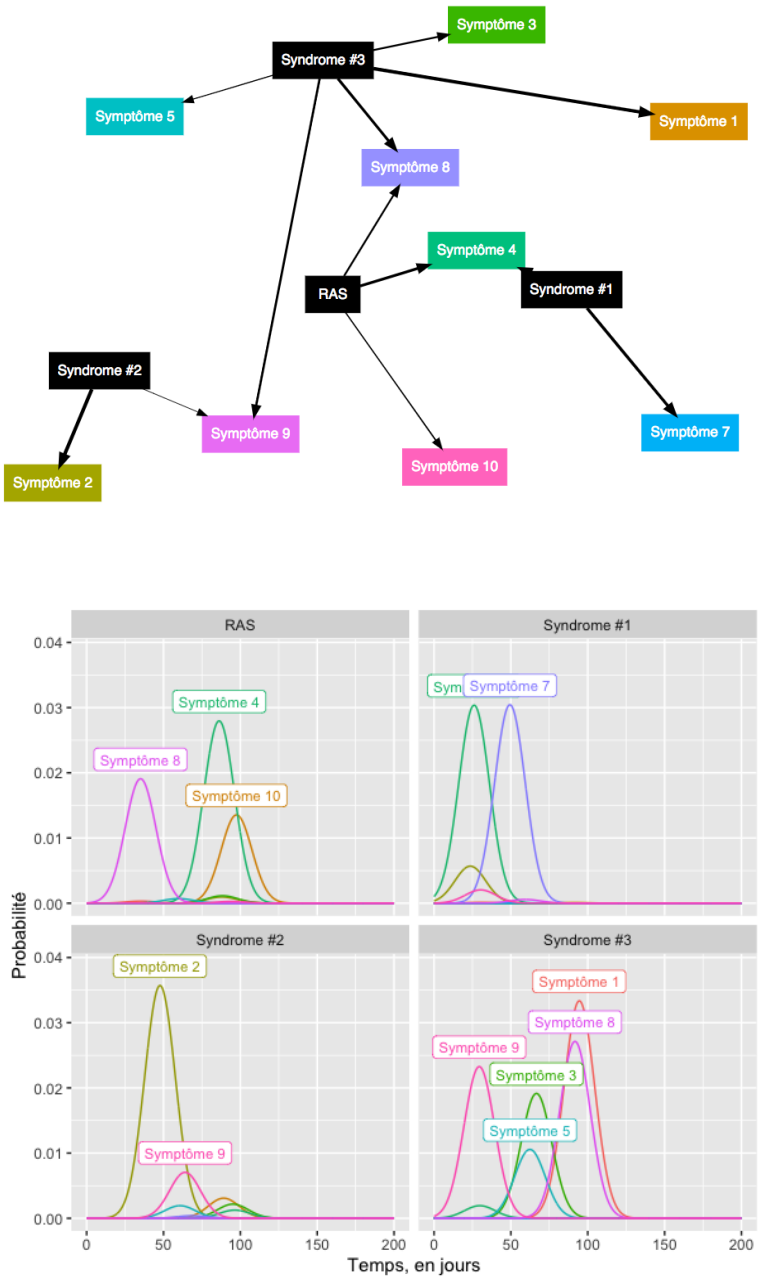

Figure 2.2: *(Top)* Symptom-syndrome relations, the thickness of the arrow indicating the probabilty that a symptom declare when the patient has the syndrome. Relations with lower than 15% probability were not traced for readability purposes. *(Bottom)* Probability of occurence, in time, of the different symptoms conditionnally to the considered syndrome, day 0 being the day wher the syndrome settles in.

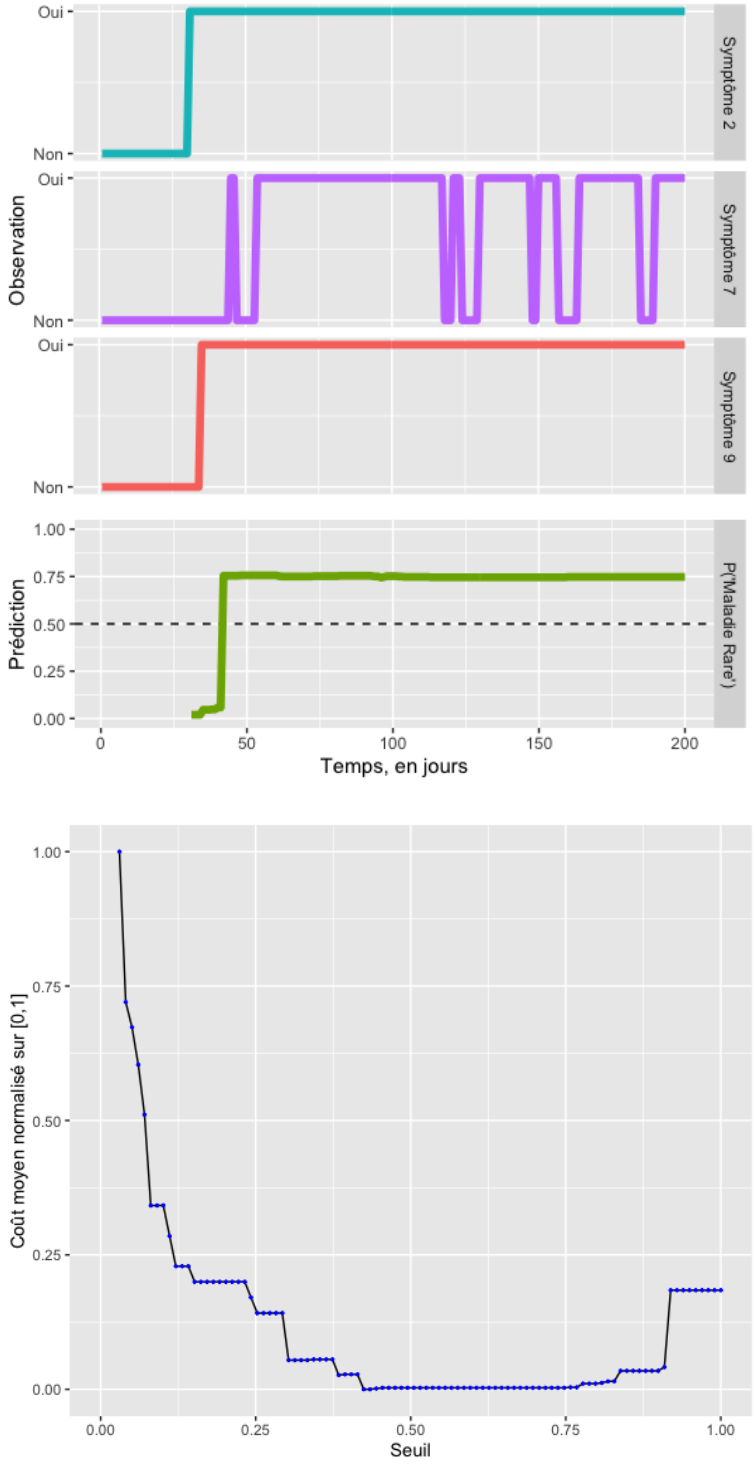

Figure 2.3: *(Top)* Data simulated over 200 days, patient has declared syndrome #1 (a rare disease) which leads often to symptoms 4 and 7, here symptoms 2 then 9 and finally 7 will be observed (first three curves), symptom 4 being present but not observed as it is latent. The prediction of rare disease risk, bottom curve in green, which started since the onset of the first symptom, symptom #2, crosses threshold 0.5 when symptom #7, which is typical for this disease is observed. *(Bottom)*. By simulating large samples of patient pathways over different syndromes, we were able to establish the average cost, as defined in section 2.2.3 conditioned on the actual threshold used to decide whether or not to send the person in a rare disease center. In this bottom graph, the cost is normalized on the [0;1] scale, since the values we used for simulation are arbitrary, hence the nominal value is irrelevant. At the extreme left, we have the cost of sending all the patients in rare disease center and at the extreme right, sending no one to the specialized center.

This work was produced jointly with Rémi Besson and Stéphanie Allassonnière, accepted at the Journées de Statistiques 2020. The code is available at github.com/FredericLoge/patientPathway.

Chapter 3

Nominations on electricity market†

Summary

In this application we consider the problem of nominations on electricity market, where suppliers announce one day ahead how much they are going to produce. Then, the suppliers get paid for the amounts announced as well as any surplus supplied. If the suppliers fail to supply their announced level however they owe money. All those prices are unavailable at the time of the decision-making, yet are at the heart of it : they correspond to the unobserved context Z in our previous setting and so, in this application, we shall focus into taking into consideration context elements in the loss function, and defining it customly of course.

3.1 Electricity markets

Let us describe briefly how electricity markets work: the central players are

- Electricity producers who sell the energy produced by their own power plants
- Suppliers who purchase electricity and sell it to customers
- Traders who buy to resell (or inversely)
- Consumers

Finally, the Transmission System Operator (TSO) is the agent responsible for ensuring the security of the power grid. Typically, they will ensure that supply and demand match, so as to avoid supply interruptions as much as possible.

In the electricity market, the notion of nomination is central: where nominating is the act of strategically announcing a quantity of energy to be sold (or bought) on the Day-ahead, based on a priori information on the Day-ahead price, forecast of electricity generation or consumption, etc. Typically, each market player makes quarter-hourly forecasts of the amount of energy it will generate or consume the next day. On this basis, these players will then make strategic nominations: they submit generation and demand offers. Two parameters define this offer: the quantity of energy and the purchase or sale price. The market is cleared once per predefined period and a single market price is determined. This is called the day-ahead or spot market. The intraday market allows the agents to readjust their positions.

Another very important phase of the electricity market is the imbalance market, where the imbalances between nominated and supplied levels are cleared. The imbalance price can be seen in two different ways:

- When the supply/demand ratio deviates, it generates costs because the system operator (TSO) must ensure the stability of the system by activating operational reserves.
- From the point of view of the direct actors of the network, a production or consumption different from the announced or necessary quantity, leads to a need to sell or buy back electricity on the market. A simple example is the case of a producer who would generate a quantity (say 2 Megawatts) greater than his forecasts. The predicted quantity is sold at the day-ahead price, and the additional 2 MW is sold on the imbalance market at a price called the imbalance price.

A simple illustration is provided in figure 3.1.

In here we focus on a simple prediction model, which would output the quantity to nominate, taking into consideration the day-ahead prices as well the imbalance prices. Those prices will not be used as features, but directly within the loss function. Note that we won't consider a storage facility in this problem and the continuous intraday market.

In the literature there has been many works, especially around Renewable Energy Systems, to find strategies for the different electricity markets, most of which model the problem as a multi-step stochastic decision problem, using stochastic programming, see [Corchero et al., 2011, Plazas et al., 2005] or reinforcement learning approaches, see [Bertrand and Papavasiliou, 2020, Boukas et al., 2020, Jiang and Powell, 2015].

Note that when working with renewables, there is increased uncertainty of production and therefore supply, hence it complexifies the bidding strategies.

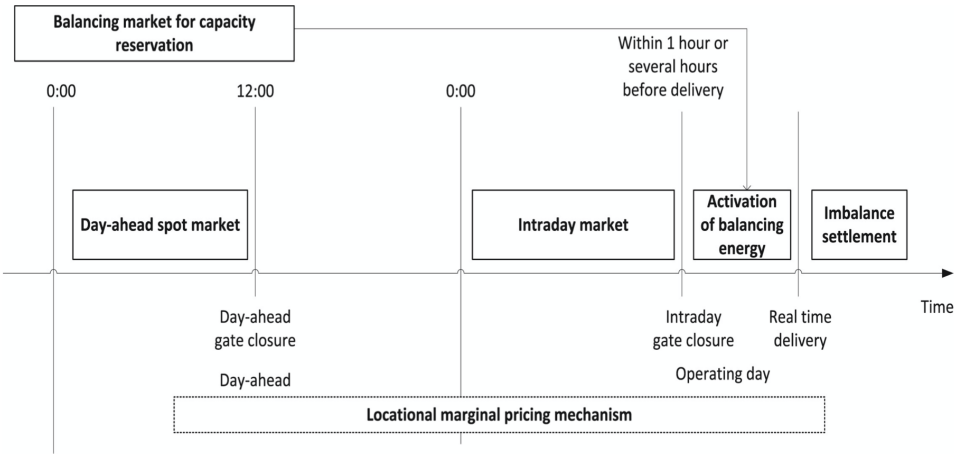

Figure 3.1: Illustration of the electric market procedure: from 0:00 to 12:00, the day-ahead spot market is opened for producers to nominate their selling quantities for the next day. The operating day, there is also an intraday market where new trade operations can be made - ignord in our applications. Once all the energy trades have been agreed, the Transmission System Operator handles the market equilibrium ("Activation of balancing energy") and the imbalance settlement is handled just afterwards.

In Meisel and Powell [2017], several instanciations of energy systems equipped with a storage device and connected to markets are studied with different algorithms to find optimal policies.

3.2 Context

We consider that we are an electricity supplier, renewable-based, connected to the Belgian grid market, for which data can be found at `elia.be`.

Every day $d-1$, before 12 am, as any supplier does, we nominate a quantity which we aim at producing for every hour of the next day. We write $(\text{nomination}_{d,h}; 1 \leq h \leq 24)$ the set of nominations announced in unit u, $(\text{production}_{d,h,q}; 1 \leq h \leq 24, 1 \leq q \leq 4)$ the set of production levels in unit u over the quarters of the next day, which of course are not known with complete certainty at day $d-1$, 12 am.

The following day, the Transmission System Operator handles the equilibrium of the market in terms of offer and demand to set the price of energy : the day ahead price and imbalance costs. The day ahead price, written $\text{day\_ahead\_price}_{d,h}$, is the energy price associated to day ahead nominations. The imbalance costs are associated to differences between the nomination and production levels: when we produce less energy than nominated, we have the obligation to buy from the market the difference, at the negative imbalance price, written $\text{imbalance\_negative}_{d,h,q}$. When we produce more energy than nominated, we can sell it at positive imbalance price, written $\text{imbalance\_positive}_{d,h,q}$.

The electric market procedure is illustrated in more details in figure 3.1.

Note that the difficulty in this problem lies in (a) the uncertainty over our production (consider renewables) and (b) the uncertainty over future prices. Otherwise, if there were no uncertainty within those elements, the decision-making problem is relatively simple and need not be framed in a data-driven perspective.

We write X_d the set of information available at day $d-1$, 12am max, in order to decide on

Name	Label	Time step
production	Our production	Quarter
production_forecast	Forecasted production, available at 12am day prior	Quarter
nomination	Nomination, decided before 12am day prior	Hour
day_ahead_price	Energy price for day ahead nominations	Hour
imbalance_positive	Imbalance price for excess energy supplied	Quarter
imbalance_negative	Imbalance cost for energy not supplied	Quarter

Table 3.1: Variables involved

nomination.

Finally, we consider the following loss function, for a chosen day d:

$$\begin{aligned}
&\mathbf{L}'\big(\hat{Y} = (\text{nomination}_{d,h})_{h=1,\dots 24}, Y = (\text{production}_{d,h,q})_{\substack{h=1,\dots 24\\ q=1,\dots 4}}, X = X_d,\\
&\quad Z = (\text{day_ahead_price}_{d,h}, \text{imbalance_negative}_{d,h,q}, \text{imbalance_positive}_{d,h,q})_{\substack{h=1,\dots 24\\ q=1,\dots 4}}\big)\\
&= -\sum_{h=1}^{24}\Bigg(\underbrace{\text{day_ahead_price}_{d,h}\text{nomination}_{d,h}}_{\text{Amount received based on nominations}}\\
&\qquad + \sum_{q=1}^{4}\underbrace{\text{imbalance_positive}_{d,h,q}\max\{\text{production}_{d,h,q} - \text{nomination}_{d,h}/4; 0\}}_{\text{Amount received based on extra-production supplied}}\\
&\qquad - \sum_{q=1}^{4}\underbrace{\text{imbalance_negative}_{d,h,q}\max\{\text{nomination}_{d,h}/4 - \text{production}_{d,h,q}; 0\}}_{\text{Amount owed based on unsupplied nominations}}\Bigg).
\end{aligned}$$

3.3 Data and working assumptions

Retrieved data We retrieved the following data from elia.be for each day d, hour h and quarter q from 2013-01-01 to 2020-02-29 :

- Wind production and day-ahead production forecast : $\text{wind\_production}_{d,h,q}, \text{wind\_production\_forecast}_{d,h,q}$
- Solar production and day-ahead production forecast : $\text{solar\_production}_{d,h,q}, \text{solar\_production\_forecast}_{d,h,q}$
- Imbalance costs : $\text{imbalance\_negative}_{d,h,q}, \text{imbalance\_positive}_{d,h,q}$
- Day-ahead price : $\text{day\_ahead\_price}_{d,h}$.

Features for decision-making We consider the following feature set, based on time information and day-ahead renewables production forecasts:

$$\begin{aligned}
X_d = \big(&(\text{wind_production_forecast}_{d,h,q}, \text{solar_production_forecast}_{d,h,q})_{\substack{1\le q\le 4\\ 1\le h\le 24}},\\
&\text{dayOfYear}(d), \text{dayOfWeek}(d)\big)
\end{aligned}$$

for day d, with dayOfYear() and dayOfWeek() extracting respectively the day of year and week from day d.

Working assumption, related to production We assume that we hold a fixed percent p of the wind production over Belgian territory, small enough such that our actions would not perturbate the dynamics of the As such, our production, for any triplet (d, h, q) is

$$\text{production}_{d,h,q} = p \cdot \text{wind\_production}_{d,h,q}.$$

Now, from the definition of the loss function, $\mathbf{L}'$ scales with the couple $(\text{nomination}, \text{production})$ i.e. for any $a \in \mathbb{R}^+$,

$$\mathbf{L}'(\hat{Y} = a \cdot \text{nomination}, Y = a \cdot \text{production}, \ldots) = a \cdot \mathbf{L}'(\hat{Y} = \text{nomination}, Y = \text{production}, \ldots).$$

As such, the value of p does not impact our analysis, as long as p is small enough that is does not impact the dynamics of the market. To simplify further results and equations, we forget this proportion, without loss of generality.

3.4 Results

Train-test split We split the available data by temporality : data strictly prior to 2018 was used as training and as test we relied on the year 2018.

Algorithmic details We first build a classic random forest, trying to predict (production_d) target variable based on available features (X_d). Practically, we relied on the R package `randomForest` to build the algorithm from the training data and the forest had the following hyperparameters :

- *ntree*, number of trees grown: 100,
- *nodesize*, minimum size of terminal nodes: 20,
- *maxnodes*, maximum number of terminal nodes: 10
- *mtry*, number of variables randomly sampled as candidates at each split: 3.

The obtained algorithm is termed afterwards "Agnostic random forest". The prediction of production level of each terminal node of each tree grown within the random forest is then recalibrated based on the protocol presented in section 1.3.2 taking as custom loss $\mathbf{L}'$. The obtained algorithm is termed afterwards "Task-driven random forest (a)". Growing completely the set of trees with loss $\mathbf{L}'$ as presented also in section 1.3.2, we obtain algorithm "Task-driven random forest (b)". All preceding algorithms provide forecasts for production, which, in our usecase, are nominations.

Baseline We consider as a baseline the case where production level for next day is known at current day and we nominate at that level. We call this the "Production-based oracle".

Result summary In table 3.2 we report summary statistics of losses $\mathbf{L}'$, in kilo euros, when we nominate based on the agnostic random forest, the task-driven random forest and the production-based oracle. Two general observations:

1. the task-driven random forest (a) manages to reach almost the average level we would have with the production-based oracle (-152.15 against -152.73 kilo euros) and highly improves

upon distribution indicators: compared to either agnostic random forest or the production-based oracle, Q1 and Q2 are better and Q3 is only slightly worse

2. the task-driven random forest (b) manages to beat the average level set by the production oracle and it is an improvement in Q1, Q2, Q3 over either the production-based oracle and agnostic random forest. However, it gets beaten by the task-driven random forest (a) over Q1 and Q2.

From an application perspective, this means that in this case one can do as good as production-based oracle with a relatively straightforward approach, as not much tuning of hyperparameters was done.

Statistic	Agnostic RF	Task-driven RF (a)	Task-driven RF (b)	Production-based oracle
Min.	-2210	**-2431**	-2236	-2186
1st Qu.	-206	**-300**	-256	-217
Median	-93	**-169**	-134	-100
3rd Qu.	-39	-35	**-44**	-40
Max.	990	3235	2221	**220**
Mean	-141.46	-152.15	**-153.24**	-152.73

Table 3.2: Summary statistics of losses, in kilo euros, rounded, when we nominate based on the agnostic random forest, the task-driven random forest and the production-based oracle. Focusing on the mean losses, we observe that task-driven random forests reach a level on-par with the production-based oracle, and even slightly better when the custom loss is used as splitting criterion.

Detailed results A natural question is the following: when the random forest was tweaked, how did the predictions change ? We answer this with the two graphs of figures 3.2 and 3.3. In the first graph, we crossed the predictions made by the agnostic random forest, the task-driven random forests and the production observed. On the second graph we crossed the losses incurred when using predictions as nominations. The predictions from the agnostic random forest correlate strongly and positively (0.93) to the true production, which is not surprising as it is what it is trained to do. However, the predictions made by the task-driven random forest (a) are strongly negatively correlated to the true production (-0.84) i.e. jointly the more we tend to produce electricity, the less we should nominate. For the task-driven random forest (b), predictions are uncorrelated to true production (-0.095). This simply takes into consideration the crucial factor: energy prices. On the graph below are plotted the losses for each of the four nomination strategies. We can observe that the loss distribution of the task-driven random forests are quite different than the others, it is typically way less asymmetric and low concentration (there is more mass on negative values)

3.5 Conclusion

As we could witness in this application, simply changing the loss function used in the trees terminal nodes allowed to recalibrate the forecasts such that the performance reached with those forecasts matched the performance reached when perfect information is used. This is an empirical demonstration of the fact that choosing a proper loss function matters much more than predicted « perfectly » the target variable (and is, conveniently, a much more attainable goal!). Please note that (a) no storage was involved and so no time dependencies are implicated here, which drastically

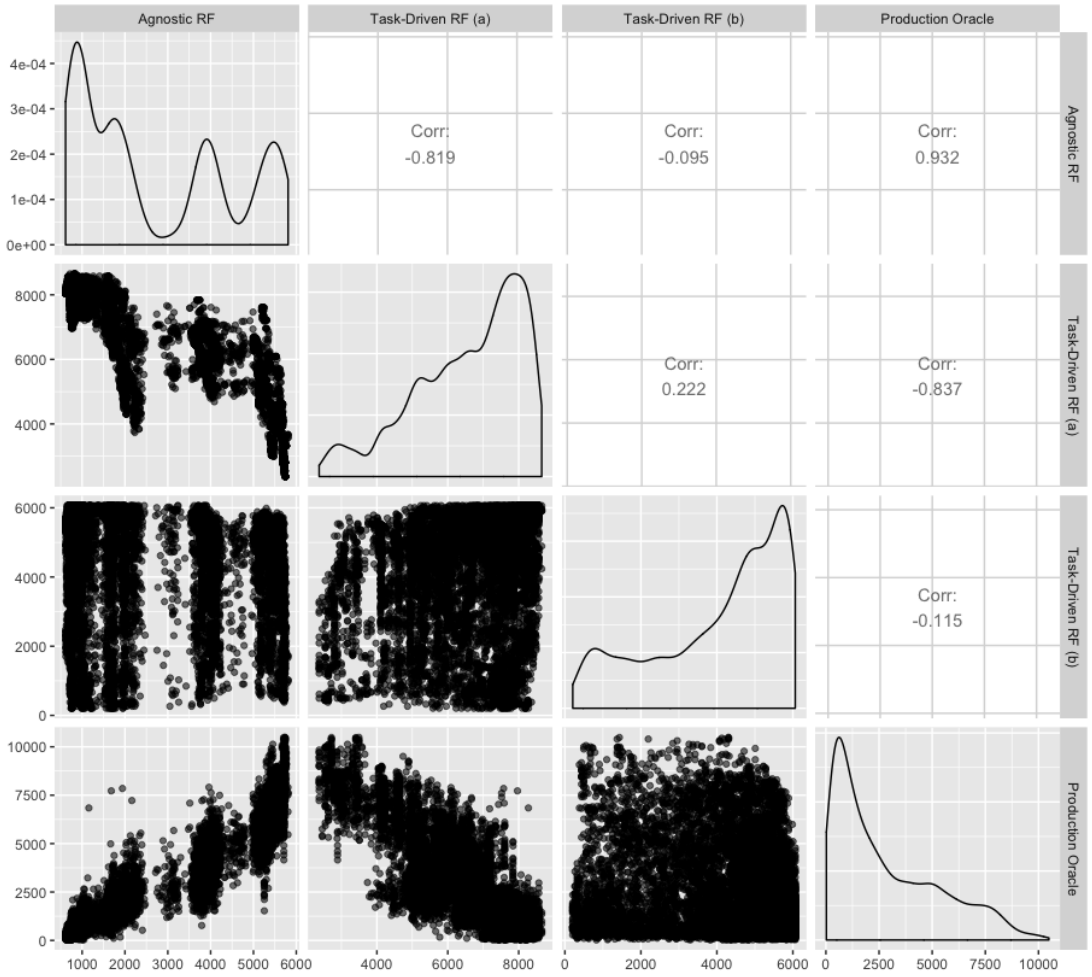

Figure 3.2: Correlation between predictions made by our three methods and the oracle: the agnostic random forest predictions are highly positively correlated with the actual production (0.932). Interestingly enough, the task-driven random forests provide very different predictions (only correlated at 0.222 between them): the first is highly negatively correlated with the actual production (-0.837) and the second is quite decorrelated (-0.115). It is even more interesting seing how the average losses observed are on-par between the two task-driven random forests-based nominations and the oracle production-based nominations.

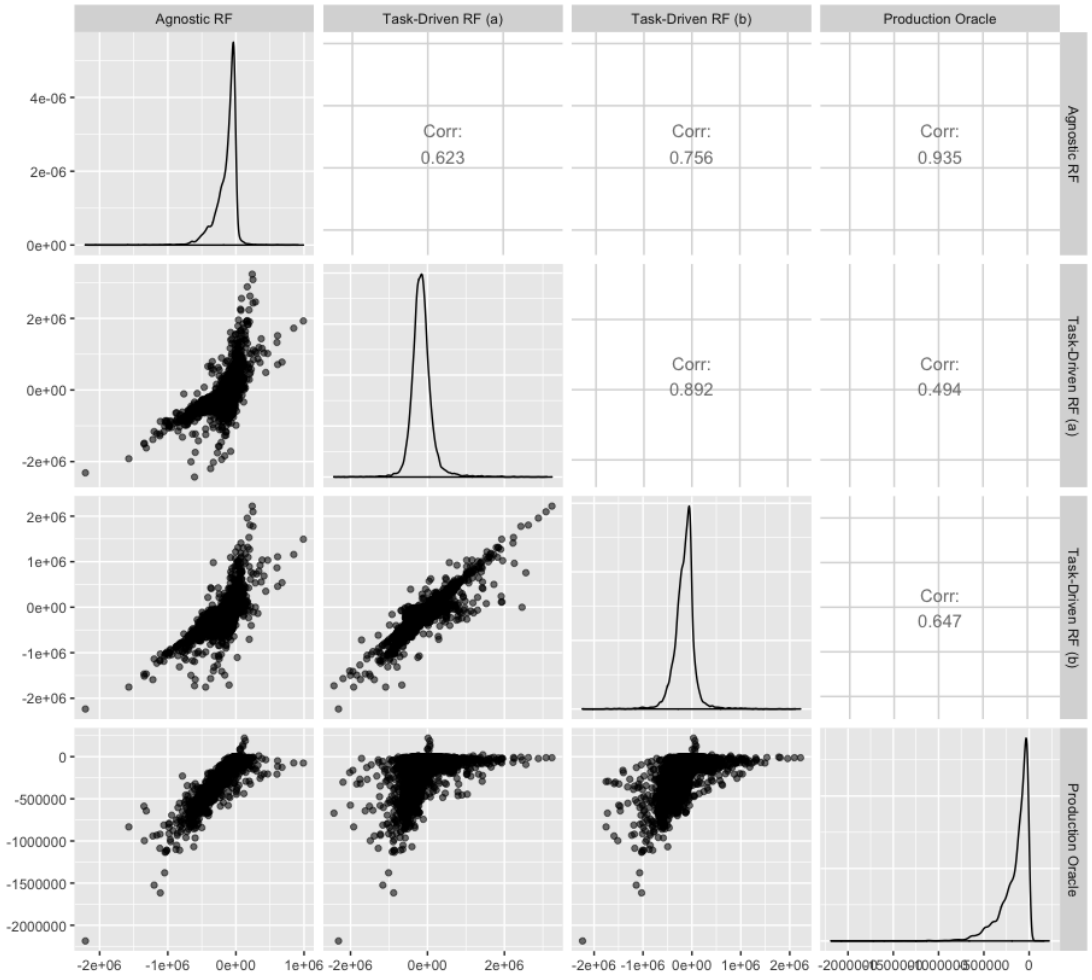

Figure 3.3: Correlation between losses made by our three methods and the oracle. We can observe that the losses between the two task-driven random forests are highly positively correlated (0.892) although the correlation between their prediction was only 0.22. We can also observe, when comparing the three random forests to the production oracle, that for the task-driven random forests, the losses tend to be more represented for negative values and for high positive values, which goes in the same way as the results presented in table 3.2.

simplify the decision problem -> deep rl would be the way to go then and (b) the optimal strategy is not to nominate your true production, although we used it as a baseline. This application allowed us also to show the flexibility and limits of our task-driven approach.

Chapter 4

Forecast-based Optimization †

Summary

In this chapter, we focus on an application where the forecast is used as input of an optimisation program, as we have introducted in section 1.2.2. We set aside contextual information (X, Z) for simplicity. Specifically, we consider the case of setting production plans one day ahead based on the forecast of customer demand, which is based on prior day demand. The problem considered is instantiated 10 times, splitted in proper train-test samples and the three approaches presented in section 1.3 were applied.

4.1 Context

In this application, we consider a production unit which is expected to supply a customer with an uncertain demand.

System details Consider a production unit, which at discrete time t:

- is operated at level $\theta_t \in [\theta_{\min}; \theta_{\max}]$, set by the operator
- takes in a quantity $I_t = I(\theta_t) = a \cdot \theta_t + b$ of energy (e.g. electricity) in order to satisfy the level of operation required by the operator
- outputs a quantity $P_t = P(I_t, \theta_t) = c \cdot I(\theta_t)$ of product (e.g. oxygen).

We hope to match with production P_t the demand Y_t of our customer (we consider an aggregated customer for simplicity), unknown at the time of the decision of θ_t. The optimal decision would be based on the actual demand Y_t, but we don't have access to it, so we will forecast this demand based on prior demands. This system is illustrated in figure 4.1.

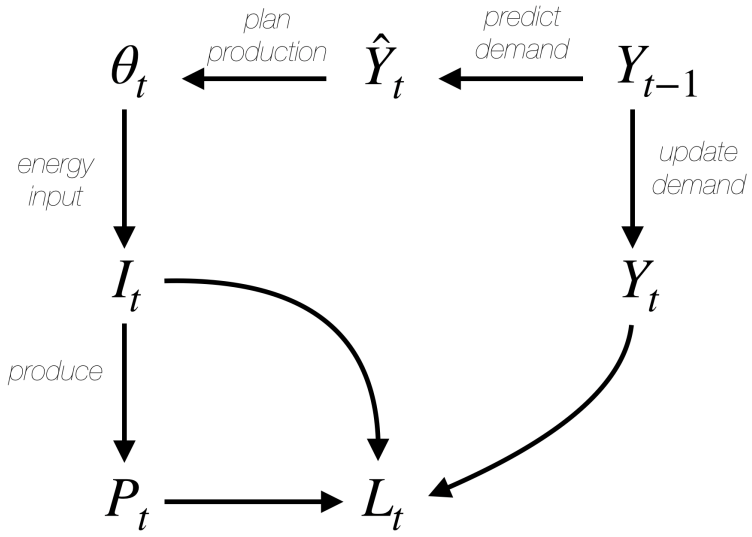

Figure 4.1: Illustration of the context considered in application 1: production plans require some forecast of customer demand, which are revealed at the end of the day only, a classic setting in real-world applications.

Cost function For control θ and demand y, we define the cost function as follows:

$$\begin{aligned} \mathbf{L}''(\theta, y, x, z) = & \; d \cdot I(\theta) && \text{Input cost} \\ & + e \cdot \max\{y - P(I(\theta), \theta); 0\} && \text{Unmet demand cost} \\ & - f \cdot \min\{y; P(I(\theta), \theta)\} && \text{Revenue.} \end{aligned}$$

Note that in this simple setting, we did not consider any observed or unobserved context (x, z respectively). Based on this definition of the cost function and prior information of $I()$ and $P()$ we have

$$\theta^*(\hat{y}, x, z) = \max\left\{\min\left\{\frac{\hat{y}/c - b}{a}; \theta_{\max}\right\}; \theta_{\min}\right\}.$$

Writing $y_{\max} = c{\cdot}(a{\cdot}\theta_{\max}{+}b)$, $y_{\min} = c{\cdot}(a{\cdot}\theta_{\min}{+}b)$ and finally bounded : $\hat{y} \mapsto \max\{\min\{\hat{y}; y_{\max}\}; y_{\min}\}$ we have

$$\mathbf{L}'(\hat{y}, y, x, z) = \begin{cases} (d/c - e - f) \cdot \text{bounded}(\hat{y}) + e \cdot y & \text{if bounded}(\hat{y}) < y \\ (d/c) \cdot \text{bounded}(\hat{y}) + (-f) \cdot y & \text{if bounded}(\hat{y}) \geq y \end{cases}.$$

This loss is represented in figure 4.2 for a given choice of (a, b, c, d, e, f).

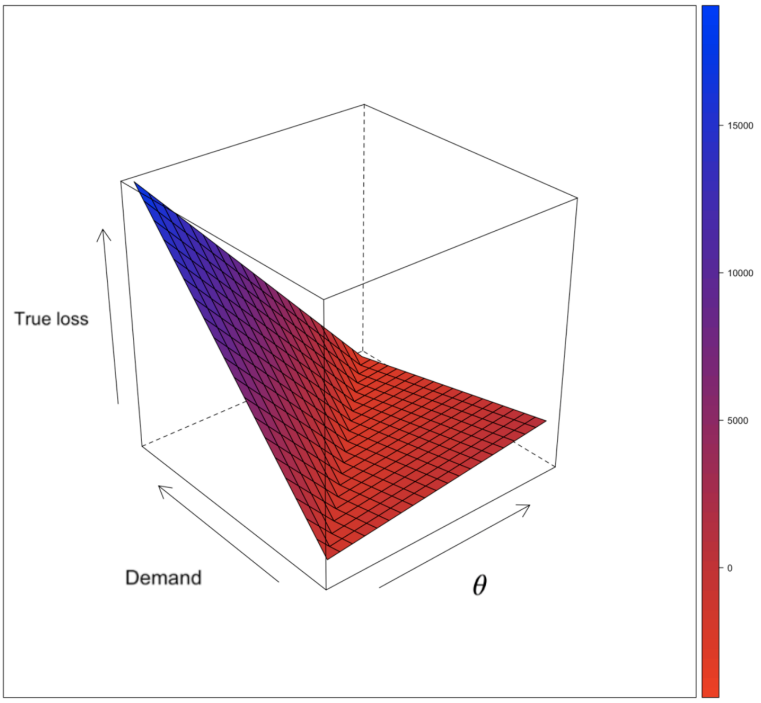

Figure 4.2: Loss function $\mathbf{L}''$ representation: loss depending on control θ and actual demand. This representation holds for our choice of parameters $(a, b, c, d, e, f) = (300, 100, 0.9, 10, 100, 20)$.

Customer demand We assume that customer demand $(Y_t)_{t\in\mathbb{N}}$ follows an Auto-Regressive model of order 1, AR(1), model with linear transformation:

$$\forall t \in \mathbb{N}, \qquad Z_t = \alpha Z_{t-1} + \sigma\varepsilon_t \tag{4.1.1}$$

$$Y_t = \beta_0 + \beta_1 Z_t, \tag{4.1.2}$$

where $\varepsilon_t | Y_t$ is a standard gaussian noise. Note that the relationship between Y_{t-1} and Y_t can be directly expressed by:

$$Y_t = \alpha Y_{t-1} + (1-\alpha)\beta_0 + \beta_1\sigma\varepsilon_t. \tag{4.1.3}$$

The AR model is a basic econometric model, justified by the idea that from one day to another, the customer demand depends linearly on itself and on external exogenous gaussian shocks in demand. It is a special case of the (S)AR(I)MA model, but a classic to start with when no seasonality and no particular trend is present.

Problem instantiation We took parameters $(a, b, c, d, e, f) = (300, 100, 0.9, 10, 100, 20)$, $\alpha = 0.8$, $(\beta_0, \beta_1) = (238.5, 10.79)$ and $\sigma^2 = 5$.

4.2 Results for tweaked decision trees

We applied different learning algorithms to predict Y_{t+1} based on Y_t: linear model (LM), CART and random forest (RF). For the task-driven approaches we computed the linear model, which was optimized using grid search analysis and is used here as baseline. We also considered the random forest with tweaked decision trees. The objective here is to demonstrate the pertinence of random forests for the task. The generative model presented in section 4.1 was simulated ten different times, 2500 observations saved for training and testing each. In figure 4.3 we plotted for each model the distribution of losses observed on the ten different instances and in figure 4.4 we plotted, for one instance picked arbitrarily, the model predictions against true values, and drawn over the points are their smooth versions to observe average behaviour.

Performances, figure 4.3 The graph of figure 4.3 shows normalized losses distributions over the ten samples, in terms of RMSE (blue) and our specific function $\mathbf{L}'$ (red, termed *Task-Driven* in the graph) for the different models considered, which are color-coded by group. We tried different versions of the algorithms, mainly either changing the optimization criterion, indicated as *Task pred*, in the case of random forests changing the splitting rule, indicated as *Task split*, and potentially adding constraints for regularization purposes, indicated by the term *constr.*. The losses are normalized by loss category, subtracting the oracle performance (which is aware of true demand) and dividing by the max of the category and we then sorted the algorithms by performance over loss $\mathbf{L}'$. From this figure, we learn three things.

(1) An overall look at the figure tells us that an unavoidable tradeoff occurs between a good performance in RMSE versus $\mathbf{L}'$; indeed one can observe that the losses correlate negatively.

(2) One may notice that, apart from the default random forest and the random forest with task prediction, all the methods with default optimization criterion (models 5-9 counting from the left) performed equally well on RMSE standard and all the methods with task specific optimization criterion (models 1-3 counting from the left) performed equally well on the Task-Driven standard. Those similar performances stand from the choice of the underlying model (linear) as well as the choice of custom function $\mathbf{L}'$.

(3) The reason the default random forest stands out is that on our model instances, the default parametrization made the random forest overfit quite a bit. As such, constraining the sample size and maximum number of nodes (relying on OOB error estimates ; models with mention *constr.*) allows us to reach similar performance even without a specific task split procedure, which implies that the splitting criterion does not impact much the performances over this specific problem. Note that we could see that the RF was overfitting as it quite poorly performed under RMSE criterion, which would seem odd in comparison with a CART algorithm.

Comparing predictions to true values Now, let us examine what actually happened to the predictions themselves, plotted in figure 4.4. We considered one of ten instances of our experiment, plotting the data points with a high transparency, because actual comparisons between the nine models simultaneously are relatively impossible on this single graph, instead we added for each model an interpolation, computed with LOESS, of the relationship between estimated and true value, and one can observe that the models gather in essentially two clusters: the three lines above the others are the methods calibrated using the custom loss, the task prediction methods. The green line, representing model "RF w/ classic split, task prediction" is slightly under all others,

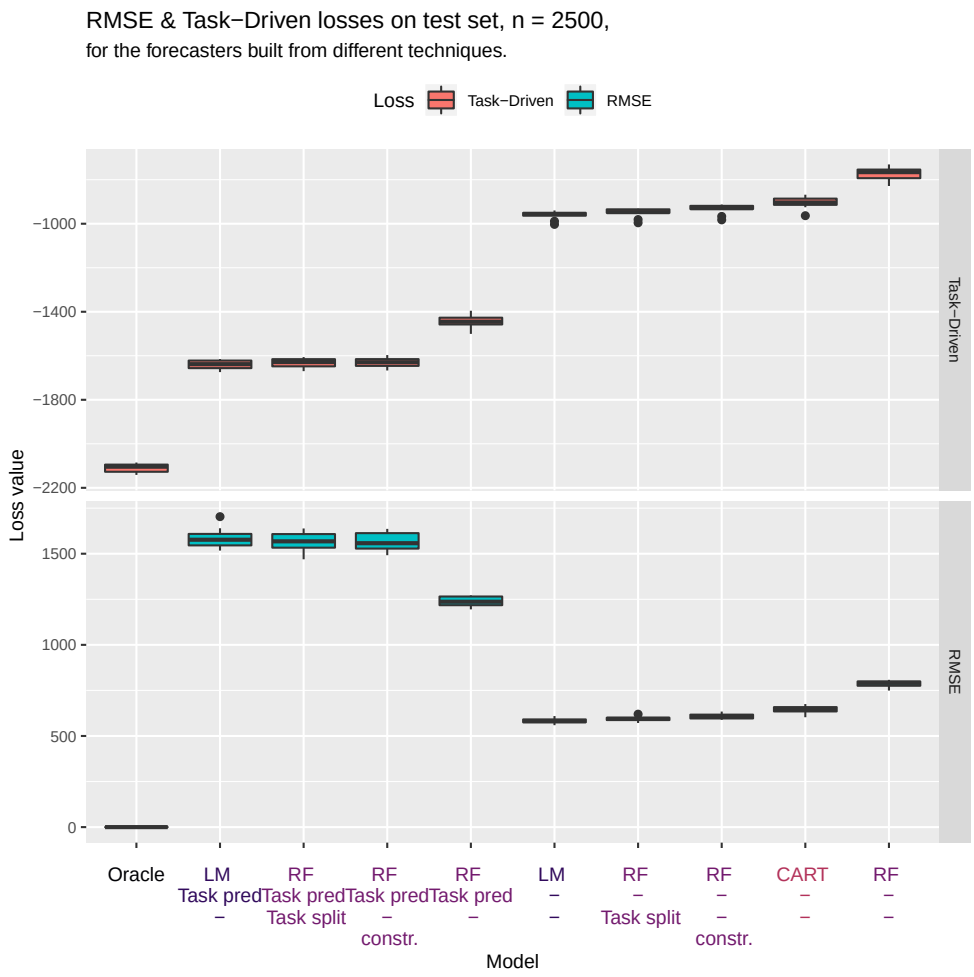

Figure 4.3: RMSE and Task-Driven losses, in blue and red respectively, for different versions of algorithms: linear model (LM), CART and random forest (RF). A color code allows the reader to distinguish easily amongst the three types of algorithm. The scores were obtained on ten replications of the experiment presented in section 4.1. To facilitate the reading of this graph, we sorted models by their average performance with respect to the Task-Driven metric, exposing the negative correlation with results obtained on RMSE metric. Please note that CART algorithm was calibrated using package `rpart` and that the random forest "RF−" was calibrated using package `randomForest` and was the basis to build "RF, Task pred -". All the other models were fully coded by us.

although it was built for task prediction as well. This model was built by recalibrating the random forest computed by function `randomForest` of the same name package, which we found out from the previous results it was largely overfitting on the data, as it had the poorest RMSE score, even worse that CART, which is unlikely if properly parametrized. Hence this forest was actually probably too deep and prone to overfitting, whatever the final prediction function used, which is why it stands between the two clusters. The second cluster, which are the lines below, gathers all the models built for the classic average squared error function. So overall, we can see that the shift from a good-RMSE to a good-Task-Driven predictor happened essentially by an average upward shift of the predictions in this particular case, which was expected from the loss function form.

4.3 Results for learned losses

Data samples, figure 4.5 Similarly to the preceding subsection, we rely on the generative model presented in section 4.1 to generate ten times train and test data batches of 2500 points each: $(Y_{t-1}, Y_t)_{t=1}^{2500}$ which let us compute the batch $\mathbf{L}'(Y_{t-1}, \hat{y})$ for our (random) selection of predictor $\hat{y}$ as presented in figure 4.5. Overall 500 data batches were built based on the initial training data, in order to explore forecast values.

Approximating the expected loss, figure 4.6 The challenge is then to approximate the expected loss from those data points, in order to find the best predictor in each case. We approximate the loss using LOESS with three different window parameters (0.02, 0.1 and 1) and a feed-forward neural network. The feed-forward neural network is composed of four hidden layers of 20 neurons each, taking as input past demand and forecast of upcoming demand, and outputing the expected loss. The neural network was calibrated using rmsprop optimizer and using early stopping based on validation set (data was half-split between training and validation). The neural network was fed all the data batches, but for LOESS, because it is so computationnally expensive, we fed it only eight data batches. As we will see afterwards, it is not so detrimental to its performance. In a particular instance out of the ten we present the predicted average loss in figure 4.6 based on the four different methods. The optimal (in this case minimal) losses found by past demand Y_{t-1} are represented by overlaying red points. One can easily notice that because of the shape of the loss function, the kernel regressor approach tends to be less satisfying than the neural network approach, because (a) its performance is overall poorer (b) the choice of bandwidth matters quite a bit. This is mostly due to the fact that the kernel is radial, and that the bandwidth choice should be adaptive to the function.

Performances, figure 4.7 The graph of figure 4.7 shows normalized losses distributions over the ten samples, in terms of RMSE (blue) and our specific function $\mathbf{L}'$ (red, termed *Task-Driven* in the graph) for the different models considered, which are color-coded by group: the kernel approach (kernel), for which different window parameters where tried, the neural network (nnet), and the linear model (classic and with task loss). The losses are normalized by loss category, subtracting the oracle performance (which is aware of true demand) and dividing by the max of the category and we then sorted the algorithms by performance over loss $\mathbf{L}'$, as previously.

The methods 2-4 counting from the left, which learnt the expected loss and then derived the optimal prediction managed to match the linear model with task loss performances "LM, Task pred". The kernel with a large window value managed to performed poorly on the Task-Driven

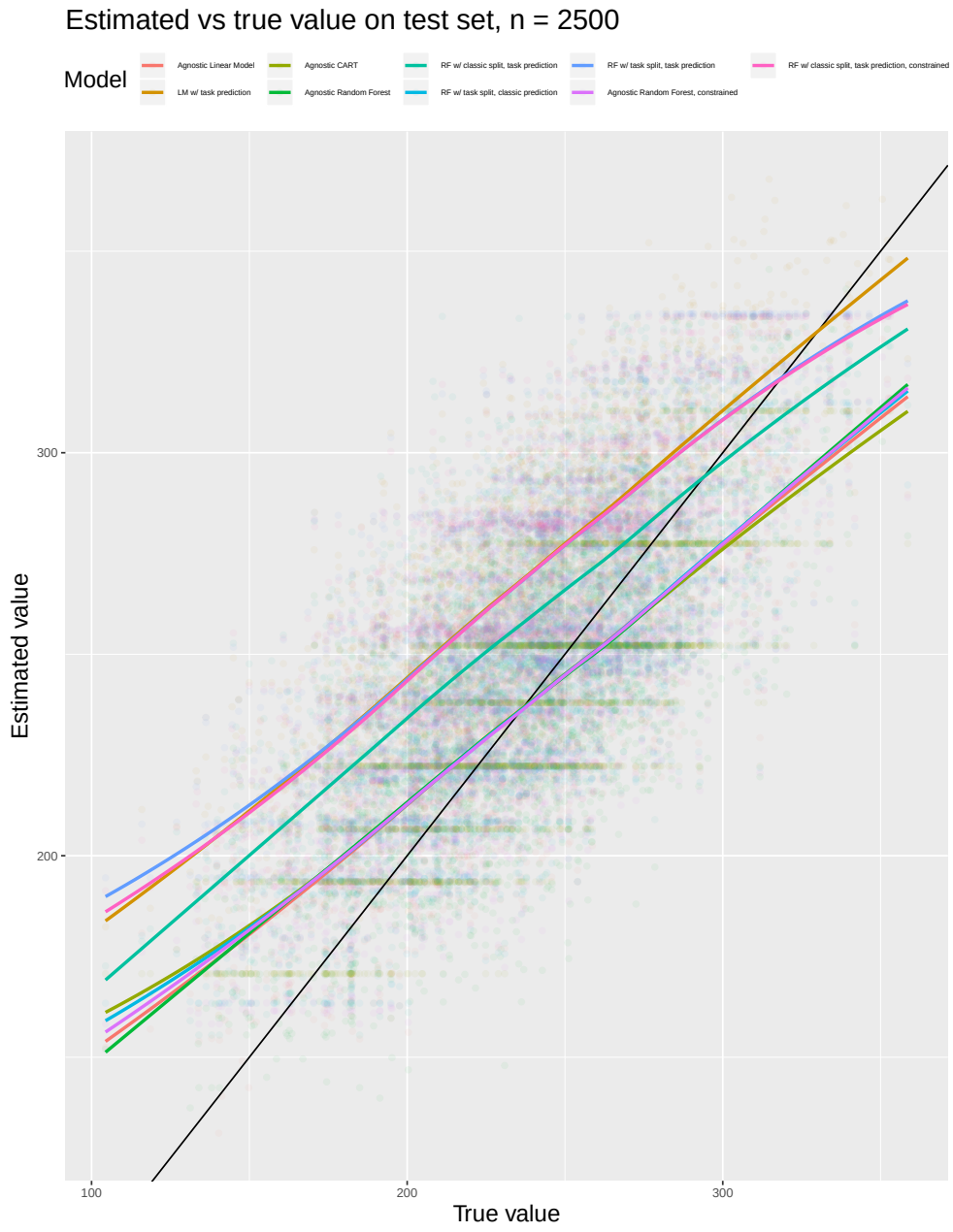

Figure 4.4: Estimated versus true values to predict, on test set of one out of the ten instances of the experiment presented in section 4.1. Because the test set is relatively large (n = 2500) and so is the number of models, we plotted the (true value, estimated value) data points with a high transparency value and added smoothed lines, to be compared with the first plane bissectrice. This graph allows us to understand how, in average, predictions were tweaked to satisfy the custom loss indicated.

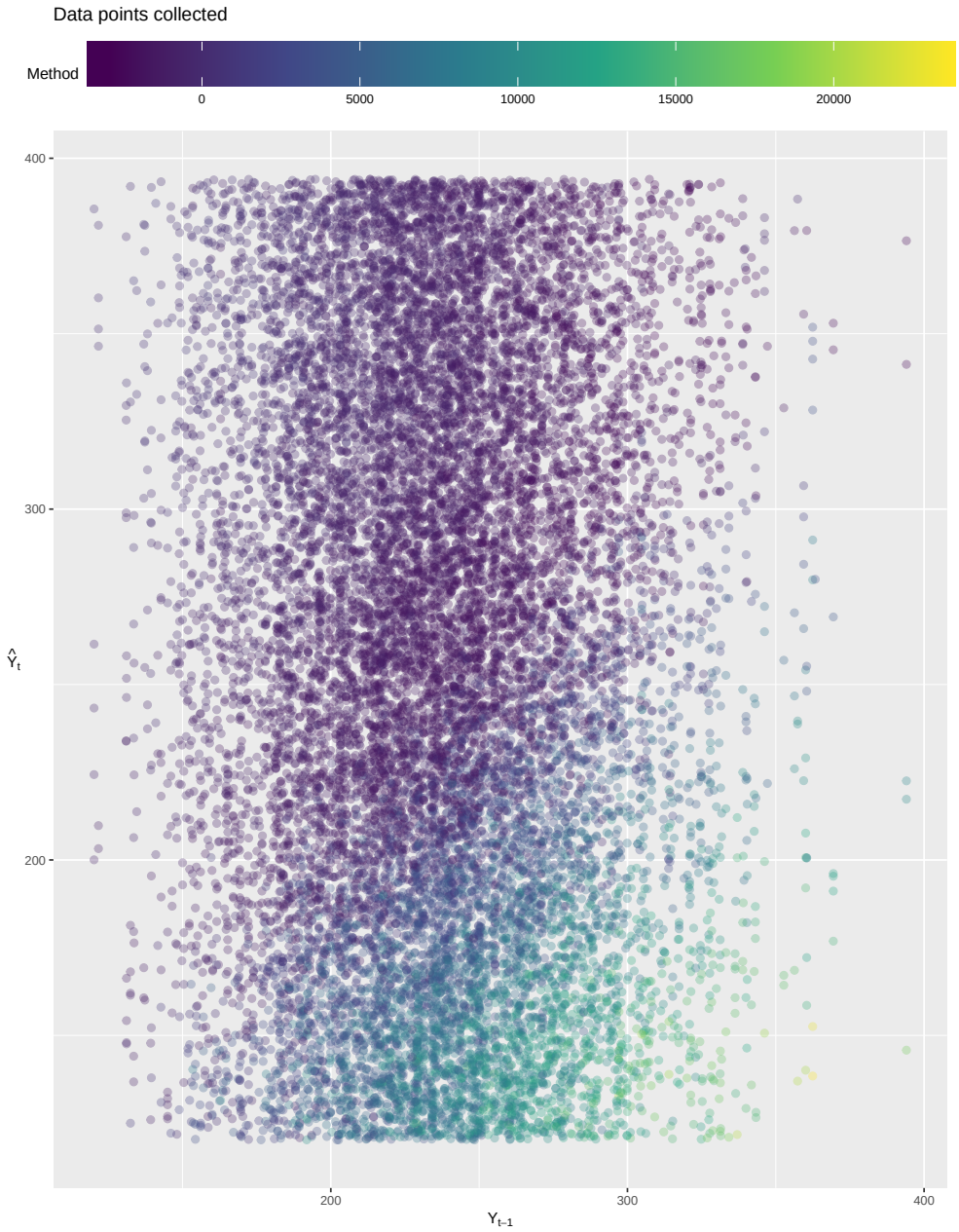

Figure 4.5: Data points $(Y_{t-1}, \hat{Y}_t^{(b)}; t = 1, \dots 2500)_{b=1}^{500}$ sampled, generating prediction $\hat{Y}_t$ under a uniform distribution over the observed range of demand: $[\min_t Y_t; \max_t Y_t]$. Data points are color-coded by the associated loss.

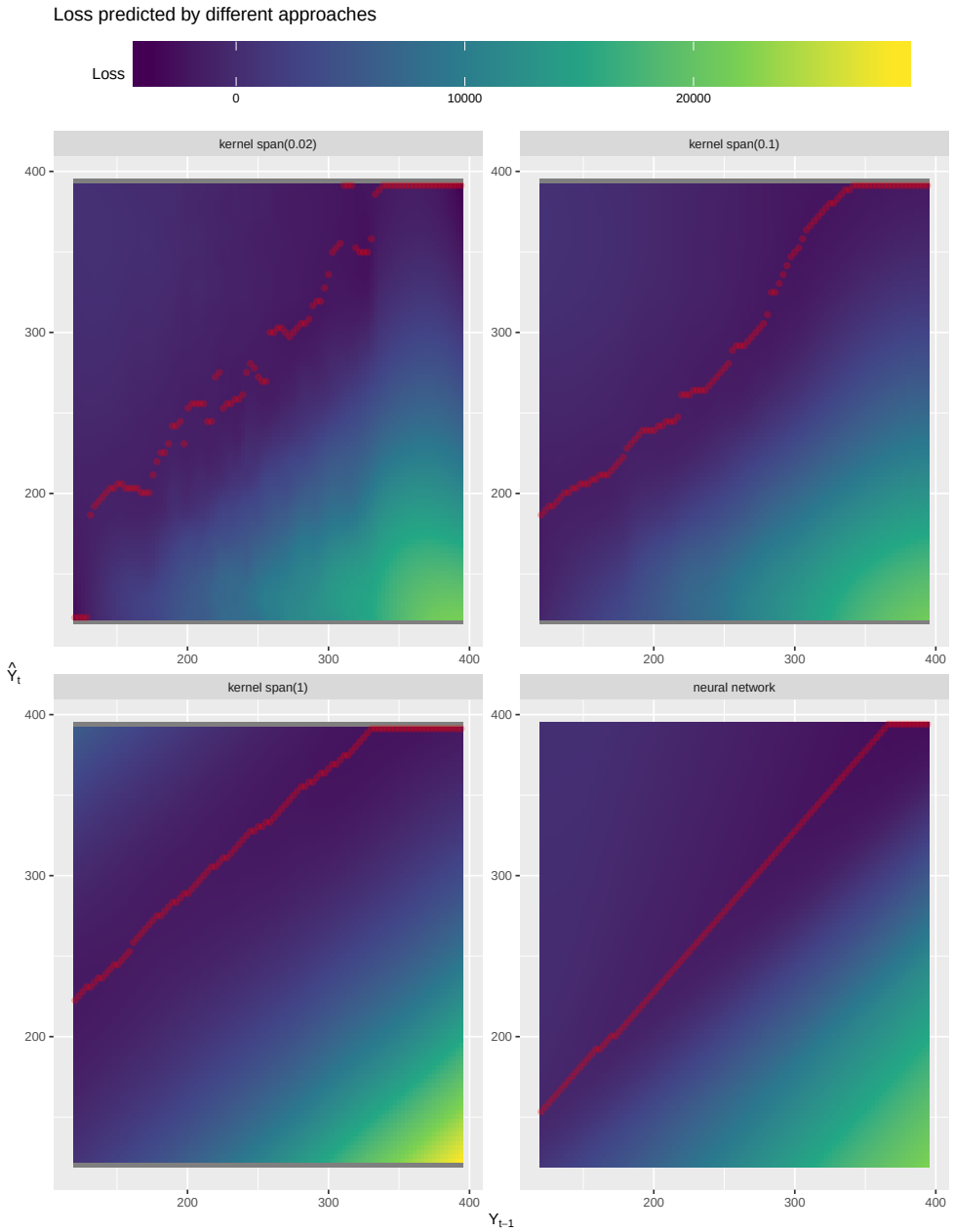

Figure 4.6: Expected loss approximated by kernels and a neural network. Note that the choice of bandwidth has, as always, significant impact on the result obtained: if too small, the map obtained clearly overfits the data points and if too large, it underfits too much, which is detrimental in performance.

loss (still better than the classic linear model) but quite worse on the RMSE loss. This shows that for an appropriate approximation of the expected loss, this method performs quite well.

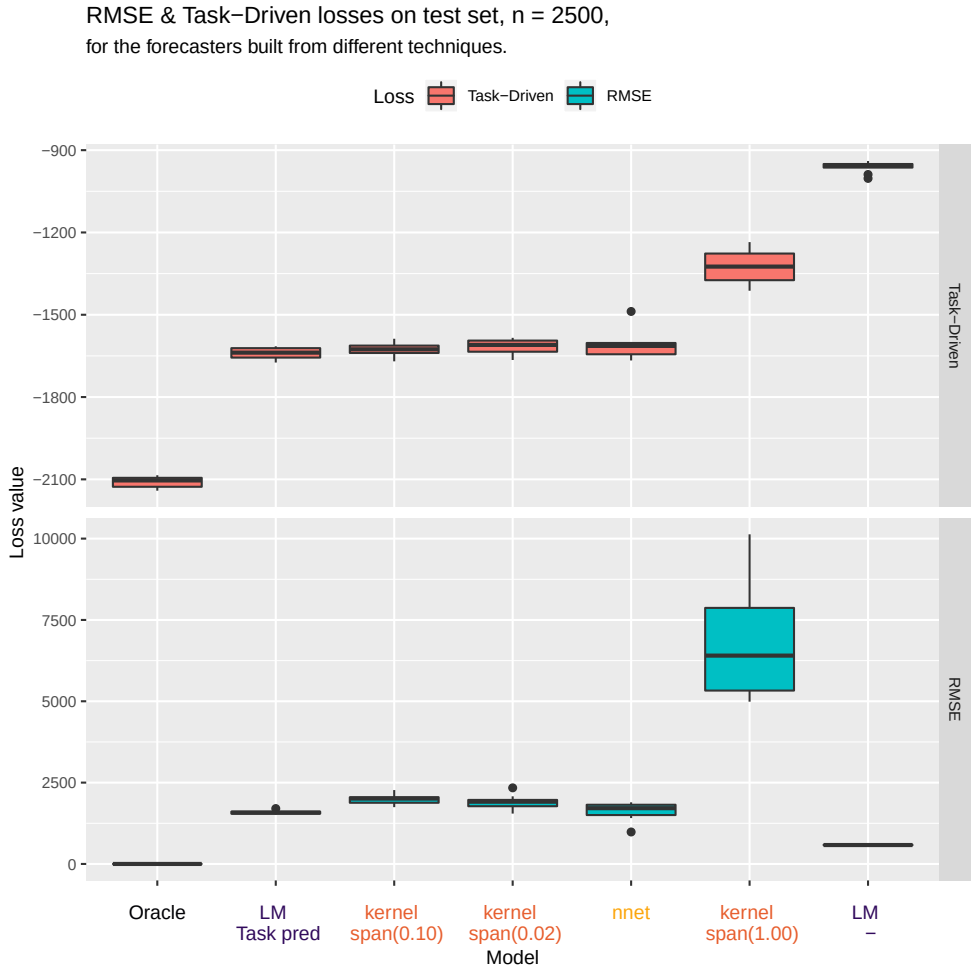

Figure 4.7: Normalized scores, RMSE and Task-Driven, in blue and red respectively, for the different approaches to learning the expected loss and thereby deriving the optimal forecast point: the kernel approach (kernel) and neural network (nnet). The linear model with classic and task prediction used in the preceding experiment served as baseline in this case. As previously, the scores were obtained on ten replications of the experiment presented in section 4.1. To facilitate the reading of this graph, we sorted models by their average performance with respect to the Task-Driven metric. The neural network was calibrated using package keras and the kernel were computed using loess method from package stats.

Comparing predictions to true values, figure 4.8 In order to complete the preceding result, we plotted the predicted vs true value observed for the different techniques tried out, adding smoothed lines of this bivariate relationship, in order to make comparisons easily. Unsurprisingly, the methods which performed similarly to the agnostic linear model behaved similarly in average, especially so for the neural network. The kernel with a large window (span = 1.0) clearly stands out of the lot, suggesting quite large predictions because it oversmoothed the expected loss function. This explains its poor performance in both RMSE and task metrics presented above. This graph is a show of proof of the validity of the second approach presented in 1.3 to solve the task-driven supervised learning problem.

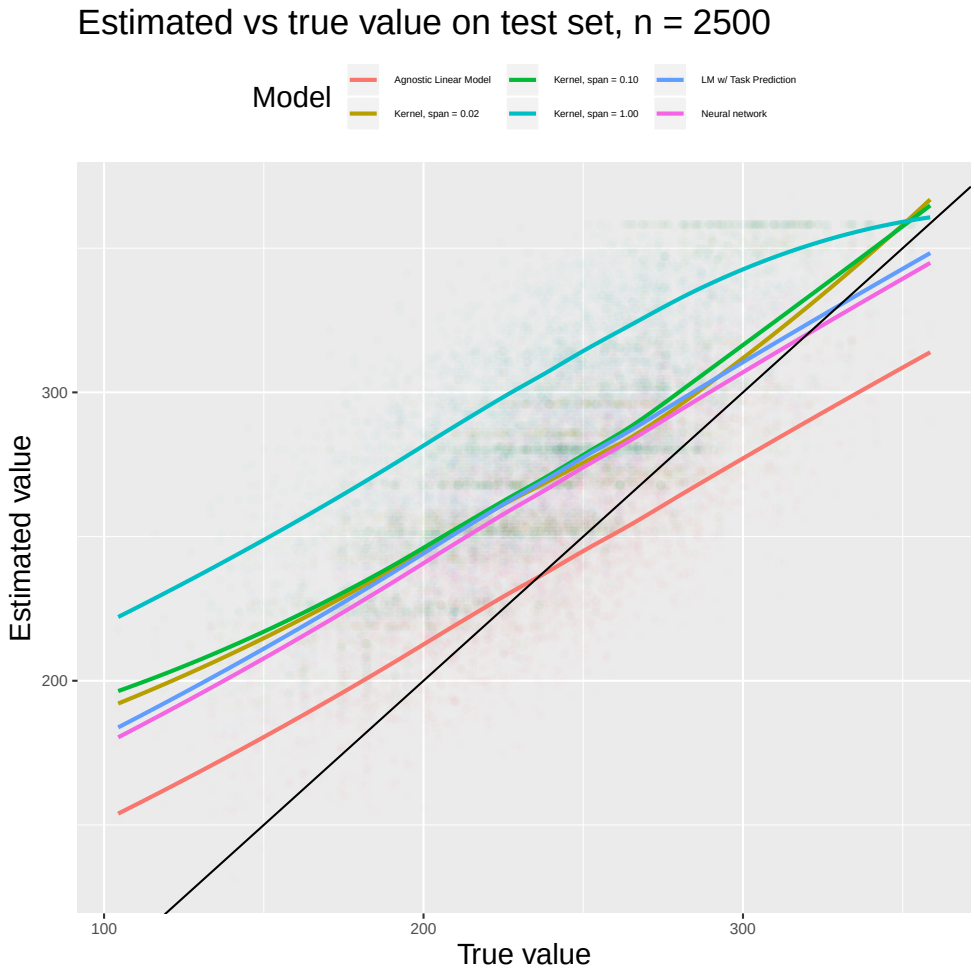

Figure 4.8: Estimated versus true values to predict, on test set of one out of the ten instances of experiment presented in section 4.1. Because the test set is relatively large (n = 2500) and so is the number of models, we plotted the (true value, estimated value) data points with a high transparency value and added smoothed lines, to be compared with the first plane bissectrice. This graph allows us to understand how, in average, predictions were tweaked to satisfy the custom loss indicated. In order to compare those results to prior experiment, we kept the agnostic and task-driven linear models as baseline elements.

4.4 Conclusion

Under a basic auto-regressive model of customer demand, we have shown how the task-driven approach we proposed earlier handles the non-quadratic loss function: basically, it considers a linear predictor from past demand, but with a much higher intercept than its true value, which helps to lower the risk of not supplying the customer, which costs much more than producing too much product and venting. Indeed: the input cost is 10 units, the revenue is of 20 units and the unmet demand cost is 100 units. Because we where in a simple setting, we where able to compute the best linear coefficients for this particular function by grid search and saw that the generic task-driven approach matched those predictors. It gives us confidence towards applying such techniques for more complex system dynamics.

Part II

Multi-step decision-making

Introduction

In the first part, the decisions taken had no future impact, while in this part, they do.

Take for instance the insulin management problem for people with type-I diabetes who self-inject insulin to regulate their Blood Glucose levels prior to meal times. Insulin dosage must take into consideration not only the current BG levels and the carbohydrate intake during upcoming meal, but also the future meals and activities planned.

Another cyclic problem is the forecast for optimisation problem, where the producer's facility has a storage unit available. As such, for a proper scaling, it is sometimes interesting to produce slightly more and store to cover for higher demands which may come later and sometimes it is interesting to produce less and rely on our storage to do the job. As a reminder, the difficulty of this problem initially was that customer demand for a given day is only revealed at the end of the day hence the storage can't be used as an immediate compensator for prediction uncertainty.

In our third application of this part, we consider the installation of a home healthcare device and follow the patients through a run-in phase. The challenge is to take proper actions at specific milestones such that the device is increasingly well tailored to the patient : ideally, the patient is comfortable, the device is adjusted and used in the best possible way. In comparison to the previous applications, this is an episodic problem, in the sense that there is a defined end of the run-in phase and we focus on a bounded time frame, whereas in the diabetes case for example we do not limit ourselves in time (even though, adjustments can and should be made along the way obviously).

Lastly, we focus on the process of acquiring data to make a specific prediction, and so we built an intelligent questionnaire, which is provided a budget of question and decides sequentially which question to ask next (based on already observed responses) in order to maximize predictive power based on a subset of the features. This work was initially motivated by an alarm system used in Healthcare, but can pretty much be applied in many different settings where user interaction can be injured by too much questions and we want to predict something. This application is episodic as well.

In all those applications, we will assume that we have at our disposal a simulator either built from medical knowledge (diabetes) or by specifying arbitrarily the environment rule specifics. We will focus on a planning approach: simulate data, model, find optimal policy. Only in the intelligent questionnaire setting will we investigate the Reinforcement Learning approach, where the question of exploration is truly brought up.

After presenting in the first subsection the basic modeling tool for such problems (Markov Decision Process) and the different algorithms to derive optimal policies we will present the four applications, starting with the two episodic problems and finishing with the cyclic problems.

Chapter 5

Markov Decision Processes and algorithms

Summary

Markov Decision Processes have become an increasingly popularized with the rise of Deep Reinforcement Learning to tackle sequential decision-making problems, as we are about to unleash in this part of the thesis. They can be seen as a quite generic framework for problems where actions are taken either at regular time intervals or at specific milestones. They represent basic agent-system interaction where an agent takes an action, the system returns some reward and some information on its state. In this chapter, we present this mathematical model and the different algorithms we are going to use in our applications to find proper decision-making approaches.

5.1 Markov Decision Process

Consider the Markov Decision Process Puterman [2014] $\mathcal{M} \triangleq (\mathcal{S}, \mathcal{A}, \mathbf{T}, \mathbf{R})$ which describes the behavior of a system's state with which we interact through actions and perceive rewards through our interactions. The state-action-reward sequence shall be written $(S_t, A_t, R_t; t \in \mathbb{N})$. The first two defining elements of $\mathcal{M}$ are respectively the state and action spaces. T denotes the transition function:

$$\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \qquad \mathbf{T}(s, a, s') = \mathbb{P}(S_1 = s' | S_0 = s, A_0 = a) \tag{5.1.1}$$

and $\mathbf{R}$ denotes the reward function:

$$\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \qquad \mathbf{R}(s, a, s') = \mathbb{E}(R_0 | S_1 = s', S_0 = s, A_0 = a). \tag{5.1.2}$$

We write Π the set of deterministic policies, where a policy $\pi \in \Pi$ is defined as a function mapping $\mathcal{S}$ to $\mathcal{A}$.

We also write $\Pi^{\text{stochastic}}$ the set of stochastic policies, where a policy $\pi \in \Pi^{\text{stochastic}}$ is defined as a function mapping $\mathcal{S} \times \mathcal{A}$ to $[0, 1]$ under the set of constraints:

$$\forall s \in \mathcal{S} \qquad \int_{a \in \mathcal{A}} \pi(s, a) = 1. \tag{5.1.3}$$

A Markovian model As the name of this class of mathematical models states, the transition kernel and reward functions are defined by conditioning on present state and actions, basically assuming that it contains the necessary information to sum up the past. Mathematically, it satisfies the Markov property:

$$\mathbb{P}(S_{t+1} | S_t, A_t) = \mathbb{P}(S_{t+1} | A_t, S_t, A_{t-1}, S_{t-1}, \ldots, A_0, S_0)$$

and

$$\mathbb{E}(R_{t+1} | S_t, A_t) = \mathbb{E}(R_{t+1} | A_t, S_t, A_{t-1}, S_{t-1}, \ldots, A_0, S_0)$$

for any $t \in \mathbb{N}$.

This assumption has the following implications. The state is the information summarizing your history in order to know what actions are available and what subsequent states you might fall in for a given action. As a consequence, it is considered as the minimum information needed for proper decision-making. Let us give some extreme examples: consider the case of the insulin management problem, where a proper prescription should actually depend on the actual level of blood glucose. If this level is omitted, what happens to the problem ? well, we would basically prescribe a good average insulin level, regardless of whether the patient is in hyper or hypo glycemia. The same thing happens when we omit important variables in the feature set in supervised learning, our predictions are simply incredibly bad. Here, we look at the consequences of such an overlook. Note that the "let's do the best with what we've got mentality" is common with Kaggle challenges and such, but this is a reminder we should be weary of the data we should retrieve.

A time-homogeneous model Although not specified in its title, we assume that throughout time, the transition and reward function is set once and for all. If time is an important factor, such as the time of the day or the year, periodic element, this element should be included in the state information, for the same reason as above. Also, we tend to look at how things evolve over a

long period of time, but it stands to reason, especially in modern times, that things change with time. As such one must be careful to choose a time window during which it is quite reasonable to state that the environment dynamics are stable. Evidently, alarm strategies can be put in place in order to check that such dynamics are stable, which we discuss in the last chapter of this thesis.

5.2 Classic examples

Let us give some classic examples of Markov Decision Processes.

Navigation problem: an episodic problem

The state represents your actual location, the actual can be a basic (north,south,west,east) or a direction on a directed graph mapping intersections of the city (Djikstra algorithm uses the same principle). In figure 5.1, we take the example of directions, where to each vertex is an associated reward indicative of the cost of taking this road with a uniform unit measure (e.g. mileage). It is common to choose one of the edges as a terminal city, to define when the tour ends. The objective is, starting in any of the edges to choose the best possible sequence of vertices to take. In this case, the problem is to be able to evaluate the best possible path out of an exponentially-growing list of sequences when the graph gets larger. There is no uncertainty with respect to the transition, so the transition kernel are dirac functions and the reward function can be stochastic if we consider for example the time taken to go from this edge to another or deterministic if we consider the number of kilometers.

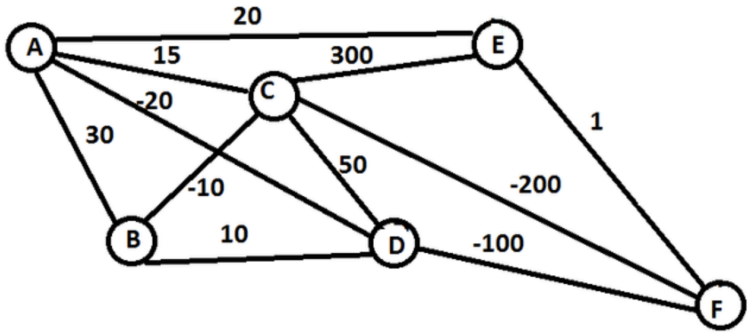

Figure 5.1: Example of MDP: mobility on MAP

The cliff problem, a classic in RL literature, is somewhat similar to the above navigation problem up to some minor changes. The scheme is represented in figure 5.2. The state still represents the physical position, the action represents the four cardinal positions. The transitions are however taken as stochastic, meaning that if we intend to go in a certain direction there is a non-zero probability we might in another. The reward function is taken as deterministic and indicative of the risk taken by the walking agent: the closer you walk to the cliff, there is more risk for you, in this case falling; instead you can take a safer path. This is interesting because it points to the problem of the multiobjective reward, combining two rewards with non-comparable scales: safety of the person, their time.

Note that both problems end at some point (not too far in time i.e. not waiting death of the agent), those are called episodic problems.

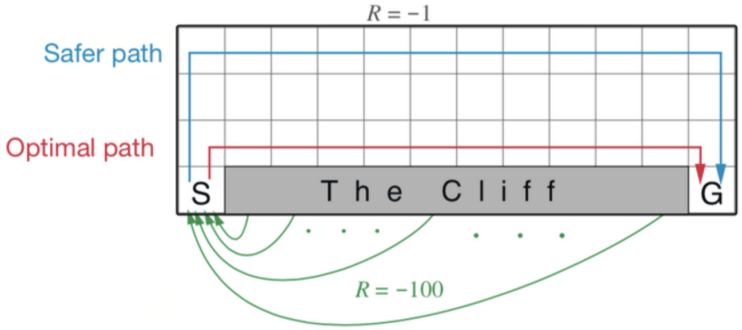

Figure 5.2: Example of MDP; mobility on cliff

Daily-life model: a cyclic problem

In figure 5.3 we propose a model for how we interact in our environment. In contrast with the previous cases, this is not episodic, as we will continuously go back to the same setting, day after day. As such we must take into consideration the long/mid-term rewards.

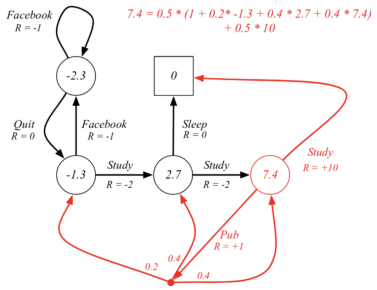

Figure 5.3: Example of MDP; daily decision-making

5.3 Value functions

Consider, for any given policy $\pi \in \Pi$, the state value function V_π which is defined as

$$\forall s \in \mathcal{S} \quad V_\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t R_t \middle| S_0 = s \right] \tag{5.3.1}$$

for any chosen $\gamma \in [0; 1]$. This γ parameter is called the discount parameter or factor, and is essentially here to ensure that the value function is truly defined and does not diverge. It can be interpreted as a parameter weighing the importance of long-term vs short term rewards, however one must not forget that from this definition, short term is always favored to long-term (equal term can be reached only in the case of episodic problems, in which case you can eventually tweak the reward as you like, typically in games where big rewards are offered when you've won or very bad if lost).

Objective: find the optimal policy

$$\forall s \in \mathcal{S} \quad \pi^*(s) \triangleq \arg\max_{\pi \in \Pi} V_\pi(s). \tag{5.3.2}$$

Several other objectives exist, such as the limit average reward, albeit less classic ; see Sutton

and Barto [2018]. The same reference provides a good overview of the different branches of Reinforcement Learning. Many communities have been working on those problems, an interesting attempt to unify domains is proposed by Warren B Powell [Reinforcement Learning and Stochastic Optimization: A unified framework for sequential decisions].

Notice the following Bellman equation: for any $s \in \mathcal{S}$,

$$\begin{aligned} V_\pi(s) &\triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t R_t \middle| S_0 = s \right] \\ &= \underbrace{\mathbb{E}_\pi \left[R_0 \middle| S_0 = s \right]}_{\text{reward at } t=0} + \gamma \underbrace{\mathbb{E}_\pi \left[V_\pi(S_1) \middle| S_0 = s \right]}_{\text{future rewards}} \\ &= \int_{a \in \mathcal{A}} \pi(s,a) \left(\int_{s' \in \mathcal{S}} \mathbf{T}(s,a,s') \mathbf{R}(s,a,s') ds' \right) da + \gamma \int_{a \in \mathcal{A}} \pi(s,a) \left(\int_{s' \in \mathcal{S}} \mathbf{T}(s,a,s') V_\pi(s') ds' \right) da. \end{aligned}$$

This recursive expression is at the root of many resolution approaches of problem stated in equation 5.3.2. The key principle here is termed *dynamic programming* and basically refers to the fact that a decision problem can be decomposed subsequently in subproblems which span over time.

Many algorithms focus on state-action value functions defined as follows: for any state-action couple $(s,a) \in \mathcal{S} \times \mathcal{A}$, we take

$$Q_\pi(s,a) \triangleq \mathbb{E}_\pi \left[\sum_{t=0}^{+\infty} \gamma^t R_t | S_0 = s, A_0 = a \right]. \tag{5.3.3}$$

5.4 Value-based methods

We take our focus on algorithms used to find an optimal policy which are based on the value functions.

We also discuss model-based elements and some basic approaches for exploitation/exploration tradeoff when we are actively using a policy, acquiring new data and refining such policy in the process. In all algorithms we will have to initially pick a discount factor γ, $\gamma \in [0,1]$, preferably not at the bounds.

5.4.1 Value-iteration algorithm

Consider a discrete (or discretized) state space and an action space which may be either continuous or discrete. By knowning/learning the transition and reward functions $(\mathbf{T}, \mathbf{R})$, one can optimize

V_π over π using the following value-iteration algorithm 3.

Algorithm 3: Value Iteration Algorithm

choose number of iterations K
input transition and reward functions $\mathbf{T}$, $\mathbf{R}$
initialize $V_0(s) = 0 \; \forall s \in \mathcal{S}$
`/* note:` V_k `denotes the estimate of the value function at iteration` k `*/`
for $k = 1, \ldots, K$ **do**
  for $s \in \mathcal{S}$ **do**
    $V_k(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathbf{T}(s, a, s')\mathbf{R}(s, a, s') + \gamma \sum_{s' \in \mathcal{S}} \mathbf{T}(s, a, s')V_{k-1}(s')$
  end
end
return $V_K(.)$

Given an estimate $V_K(.)$ of the state value function $V_{\pi^*}(.)$, our approximation of the optimal policy is given following:

$$\forall s \in \mathcal{S} \quad \pi^*(s) \approx \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathbf{T}(s, a, s')\mathbf{R}(s, a, s') + \gamma \sum_{s' \in \mathcal{S}} \mathbf{T}(s, a, s')V_K(s'). \tag{5.4.1}$$

5.4.2 Q-Learning algorithm

In the case of discrete state and action spaces, and assuming we have observed a set of state-action-state-reward sets, we can estimate the state-action value functions using the tabular Q-learning algorithm 4.

Algorithm 4: Q-Learning Algorithm, tabular case

initialize $Q(s, a) = 0 \; \forall s \in \mathcal{S}, a \in \mathcal{A}$
input state-action-state-reward sets $(S_t, A_t, S_{t+1}, R_t; t = 1, \ldots, T)$
for $t = 1, \ldots, T$ **do**
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \eta \left(R_t + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right)$
end
return $Q(.,.)$

With sufficiently enough data processed, the optimal policy is approximated as follows:

$$\forall s \in \mathcal{S} \quad \pi^*(s) \approx \arg\max_{a \in \mathcal{A}} Q(s, a). \tag{5.4.2}$$

It is standard practice to provide in input a large number of independent sequences of state-action-state-reward rather than a single trajectory, as this method requires theoretically infinite exploration in terms of space and action spaces to provide convergence guarantees.

Now, suppose the state is continuous or high-dimensional, in which case the previous tabular case does not properly treat the variable or is in practice difficult: the previous update rule must be modified. Consider that the state-value function is actually parametrized by vector w (e.g. neural network, linear basis), which we then write Q_w. The corresponding algorithm 5 updates

the weight with new data batches.

Algorithm 5: Batch Parametric updates

initialize w_0 at random
pick learning rate η
choose number of iterations K
for $k = 1, \dots, K$ **do**
  Collect $\mathbf{batch}_k$ based on rule: $A_t = \arg\max_{a \in \mathcal{A}} Q_{w_{k-1}}(S_t, a)$
  Update $w_k \leftarrow w_{k-1} + \eta \frac{\partial}{\partial w} \left[\sum_{t \in \mathbf{batch}_k} \left(R_t + \gamma Q_w(S_{t+1}, A_{t+1}) - Q_w(S_t, A_t) \right)^2 \right]$
end
return w_K

Given an estimate of state-action value function Q_{w_K}, we have the following approximation for the best possible action:

$$\forall s \in \mathcal{S} \quad \pi^*(s) \approx \arg \max_{a \in \mathcal{A}} Q_{w_K}(s, a). \tag{5.4.3}$$

In some cases, it might be a good idea to freeze for a certain number of iterations the current state-action value function estimate so as to not make this estimate diverge. This is a problem typically referenced under the *deadly triad* term, making reference to the three elements which, jointly lead to learning instabilities: bootstrapping, off-policy approach, function approximation. Using a frozen set stabilizes target value and applying replay memory breaks temporal dependencies between transitions. This strategy has shown good results for instance in Mnih et al. [2013] where Deep Neural Networks were used as function approximators for Atari game controllers.

5.4.3 Backward algorithm

If states are somehow hierarchical and episodic (depth K) e.g. tic-tac-toe problem, we can iterately learn the optimal value functions by starting with the last states, then the second to last states, then the third to last states and so on and so forth.

We write $\mathcal{S}_{\text{layer}:k}$ the space of states of layer k, for any $k \in \{1, \dots, K\}$.

Specifically, the procedure starts by solving

$$V_{\pi^*}(s_K) = \max_{a_K \in \mathcal{A}} \mathbb{E}\left[R_K | S_K = s_K, A_K = a_K\right] \tag{5.4.4}$$

and sequentially solve, for $k \in \{K-1, \dots, 0\}$

$$V_{\pi^*}(s_k) = \max_{a_k \in \mathcal{A}} \mathbb{E}\left[R_k + V_{\pi^*}(S_{k+1}) | S_k = s_k, A_k = a_k\right], \tag{5.4.5}$$

substituting sequentially estimates of value functions for the following layer. In this case, one can either directly solve those optimization problems via function approximation or by estimating model components.

Algorithm 6 properly describes the process properly. Note that functions $(f_k)_{k=1}^K$ represent

state-action value functions, indexed by layers.

Algorithm 6: Learning value functions

learn f_K as $\hat{f}_K$, where

$$f_K : \mathcal{S}_{\text{layer:}K} \times \mathcal{A} \to \mathbb{R}$$
$$(s, a) \mapsto \mathbb{E}[R_K | S_K = s, A_K = a]$$

for $k \in \{K-1, K-2, \dots, 1\}$ **do**

  learn f_k as $\hat{f}_k$, where

$$f_k : \mathcal{S}_{\text{layer:}k} \times \mathcal{A} \to \mathbb{R}$$
$$(s, a) \mapsto \mathbb{E}\left[R_k + \max_{a' \in \mathcal{A}} \hat{f}_{k+1}(S_{k+1}, a') | S_k = s, A_k = a\right]$$

end

return set of networks : $\{\hat{f}_k, k \in \{1, \dots, K\}\}$

Based on the set of networks, we have, for any layer $k \in \{1, \dots, K\}$:

$$\forall s \in \mathcal{S}_{\text{layer:}k} \quad \pi^*(s) \approx \max_{a \in \mathcal{A}} \hat{f}_k(s, a). \tag{5.4.6}$$

5.5 Model-based methods

We provide here estimates of the transition and reward functions, stated in equations 5.1.1 and 5.1.2. Although in practice we often have access to multiple state-action-reward sequences, we formulate the estimates with a single sequence, which keeps notations simple, without losing much generality. This single sequence is written with initial notations $(S_t, A_t, R_t; t \in \{1, \dots, T\})$.

5.5.1 Discrete case

Assuming that state and action spaces are discrete, and that the number of transitions observed $T-1$ is sufficiently large (and with sufficient diversity) with respect to $\mathbf{card}\{\mathcal{S} \times \mathcal{A}\}$, one can use a counting estimate for transitions and a average estimate for rewards. Consider a state-action-state set $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, the transition estimate would write:

$$\hat{\mathbf{T}}(s, a, s') = \frac{\mathbf{card}\{t \in \{1, \dots, T-1\} : S_{t+1} = s', S_t = s, A_t = a\}}{\mathbf{card}\{t \in \{1, \dots, T-1\} : S_t = s, A_t = a\}} \tag{5.5.1}$$

and the reward estimate would write

$$\mathbf{R}(s, a, s') = \frac{\sum_{t \in \{1, \dots, T-1\}} R_t \mathbb{1}\{S_{t+1} = s', S_t = s, A_t = a\}}{\sum_{t \in \{1, \dots, T-1\}} \mathbb{1}\{S_{t+1} = s', S_t = s, A_t = a\}}. \tag{5.5.2}$$

Such estimates, albeit straightforward suffer three linked disadvantages.

First, in a non-discrete setting, they would require the user to choose a discretization of the state and action spaces. This is a non-trivial task as, with more precise states, you get closer to the optimum model specification and therefore finding the optimal policy, but you would certainly need a lot of data to make this very fine grid work. On the other hand if you take an imprecise grid,

you will surely have enough data to produce precise enough estimates, but those will be highly biased, and therefore lead to suboptimal policies.

Second, each cell of the grid is considered somewhat independently to the others. Indeed, the model estimates for two cells (even adjacent) come from different data points. Surely there is a way to leverage points on adjacent cells to make more precise estimates.

Third, rarely visited states suffer from poor estimates compared to others. Hence the grid should actually depend on the behavior of the MDP if we want to estimate things in a homogeneous ways (adaptive estimates in non-parametric regression). Augmenting the number of simulations is not always sufficient e.g. if the state is a gaussian variable, then the extremes are always rarely visited, whatever the number of observations is.

Now, a parametric, or at least semi-parametric, approach can help alleviate those issues, as the structure hypothesized provides better convergence rates than any non-parametric approach presented above.

Another classic extension of prior model estimates is to bring structure by relying on kernels and neighborhood distances in order to quantify similarities between different states and between different actions, such that when an observation is acquired, it brings information not only to a specific state-action-state set, but to several ones.

5.5.2 Continuous case

Consider now that state and action spaces can be continuous. In this case, the kernel and neighborhood function approach makes even much more sense than in the discrete setting.

Let us consider the transition estimate for example. Let $\mathbf{N}_{\mathcal{A}}$ and $\mathbf{N}_{\mathcal{S}}$ denote two neighborhood measures which compare pairs of actions and states respectively. Now, for state-action-state set $(s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, we would take the following estimate of the transition probability:

$$\hat{\mathbf{T}}(s, a, s') = \frac{\sum_{t \in \{1,\dots,T-1\}} \mathbf{N}_{\mathcal{S}}(S_{t+1}, s')\mathbf{N}_{\mathcal{A}}(A_t, a)\mathbf{N}_{\mathcal{S}}(S_t, s)}{\sum_{t \in \{1,\dots,T-1\}} \mathbf{N}_{\mathcal{A}}(A_t, a)\mathbf{N}_{\mathcal{S}}(S_t, s)}. \tag{5.5.3}$$

As was pointed out previously, instance-based approaches, such as kernels and neighborhood functions suffer from the curse of dimensionality: the amount of data required grows exponentially with the dimension and in particular, point-to-point comparisons in a high-dimension space become meaningless since no point is particularly far or close to another. Thus, in the case of high-dimension, one might want to impose some structure via (semi-)parametric models, so that estimates are reliable.

5.6 Outside pure value-based methods

Previously, we have focused mainly on value-based (improved sometimes with model knowledge so model-based too) approaches.

Another approach to find the optimal policy is to actually directly optimize the policy: policy-based optimization, Pontryagin principle. The most famous algorithm to do so is REINFORCE, see Sutton and Barto [2018]. The idea is to parametrize the policy and take the gradient over its parameter vector, and perform gradient descent.

A few years ago, was proposed the idea to merge the two approaches of value and policy-based methods to make a better algorithmic procedure: there was born the Actor-Critic.

Around the same time there was the emergence of using neural networks as value-function approximation, which helped tremendously for, at the time, complex tasks such as Atari Games, game of Go, StarCraft, etc.

5.7 Going online

Let us assume that we are at iteration t of the algorithm, we dispose of estimates of the Q value function, written $\hat{Q}_t$. We write s the current state and A the action we are going to pick. Let us give a few adaptations of the strategies we have seen with bandits.

No exploration Note that playing greedy strategy is

$$A|s = \arg\max_{a \in \mathcal{A}} \hat{Q}_t(s, a). \tag{5.7.1}$$

Uniform The simpler exploration strategy, is to pick at random, without any apriori a point in the action space

$$A|s \sim \text{Uniform}(\mathcal{A}(s)). \tag{5.7.2}$$

where we wrote $\mathcal{A}(s)$ as the action space acceptable for state s. Note that the famous ε-greedy strategy consists in splitting the iteration times between not exploring and picking at random, which is a legitimate approach.

Softmax Consider the action space discrete. There is the softmax strategy, which consists in taking

$$A|s \sim \text{Multinomial}\left(\frac{\hat{p}(\beta)}{||\hat{p}(\beta)||_1^1}\right) \tag{5.7.3}$$

where vector $\hat{p}(\beta)$ is defined as the following set of pseudo-probabilities:

$$\hat{p}(\beta) \triangleq \left\{\forall a \in \mathcal{A}, \exp\{\beta \hat{Q}_t(s, a)\}\right\} \tag{5.7.4}$$

with $\beta \in \mathbb{R}_+$ the softmax parameter. Depending on the value chosen for β, the strategy ranges from picking actions uniformly at random, case $\beta = 0$, to the hardmax / greedy strategy, with $\beta \to +\infty$.

In the continuous action space case, we can take similarly

$$\mathbb{P}(A|s) \propto \exp\{\beta \hat{Q}_t(s, .)\}. \tag{5.7.5}$$

The clear advantage of this technique is that, for a reasonable choice of β, we do not explore on poorly performing state-action couples, but we do not center too much on best ones, unless they very much distinguish themselves from the lot. However, it must be noted that estimate uncertainty was not taken into account here. Which leads us to the Upper-Confidence Bound approach (UCB).

Upper-Confidence Bound (UCB) Consider a discrete state-action space and let $n_t(s, a)$ be a counting function of the number of cases where action a was taken in state s. Then, the UCB criterion is defined as

$$\hat{Q}_t(s, a) + \alpha B(n_t(s, a), t) \tag{5.7.6}$$

where α is positive and indicates the balance between exploitation and exploration, as function B is decreasing with its first argument and increases with its second one. A classic choice is the following:

$$B(n_t(s,a),t) = \sqrt{\frac{\log(1+t)}{n_t(s,a)}}. \tag{5.7.7}$$

The $n^{-1/2}$ is the classic estimation error bound in statistical learning, and the log term helps slowly increasing exploration with time, in case anything was missed.

Note that this approach, contrary to the softmax, is deterministic, hence might lead to less rapid exploration, but this largely depends on the parameter choice, and at least when UCB discards an action it is because it has been tested enough to be sure that it is not a potential best. Please note also that one can combine the two approaches by substituting in the softmax pseudo-probabilities the estimate by their upper-confidence bounds, in order to take into account our uncertainty upon it.

Bayesian Closely related to UCB and softmax, the idea is that we have a prior on the distribution of Q-values for example and we update it with data. Rather than picking some maximum, we draw from each action posterior distribution and then pick the action with maximum output. Evidently, the distributions which are superior, in the stochastic sense for example, will be more likely to be chosen.

Chapter 6

Forecast-based Optimization, involving a storage unit †

Summary

In chapter 4, we considered a production unit requiring us to predict customer demand based on prior days. In this chapter we extend this application by adding a storage unit to the production site, so that it may be used in low-demand days as a substitute for direct production, and in high-demand days, where demand actually exceeds production capacity of the site.
Two approaches are considered to tackle this problem: either optimizing the demand forecast which comes as input of the proper optimisation program handling both production and storage control or optimizing directly production and storage based on the prior day demand.

6.1 Context

In this application, we consider a production unit, linked to a storage facility, which is expected to supply a customer with an uncertain demand.

6.1.1 Bibliographical elements

Equipping a production unit with a storage device allows more flexibility: we can supply more at a given time by unstoring, we can take advantage of variability in energy prices (when price is quite low, tend to produce more and store; when price is quite high, tend to produce less and unswore to fulfill supply). Of course, storing and unstoring comes at some cost. In any case, those possible adjustments require a proper optimization problem. Note that in the energy sector, storage devices are particularly interesting to handle uncertainties related to renewable generation.

In most cases, a clear knowledge of the system dynamics (storage include) is required to solve this problem. Let us give a few examples. In Wang et al. [2009], authors study a Reservoir Management problem combining in a smart way system information (model-based approach) and data assimilation. In Sarma et al. [2006], the authors formulate the production optimization problem following a constrained nonlinear programming problem, with the constraints computed following a model for the system dynamics. In Haijema et al. [2007], the authors use MDP model and simulation to provide simple rules to handle blood platelet production and storage, which are precious and perishable.

Finally, the difficulty of using forecasts as input of an optimisation program has been studied in the references mentioned in section 1.2.2.

6.1.2 Details

Let us extend the context presented in chapter 4.

Consider a supplier, which at discrete time t:

- chooses a production level $\theta_t^{prod} \in \left[\theta_{\min}^{prod}; \theta_{\max}^{prod}\right]$
- buys input quantity $I_t = I(\theta_t^{prod}) = a \cdot \theta_t^{prod} + b$ of energy (e.g. electricity) in order to satisfy the level of production θ_t^{prod} decided upon
- produces a quantity $P_t = P(I_t, \theta_t^{prod}) = c \cdot I(\theta_t^{prod})$ of product (e.g. oxygen)
- chooses a storage plan $\theta_t^{store} \in \{\texttt{store}, \texttt{unstore}, \texttt{none}\}$ such that:
  - if `none` is selected, the customer is supplied all the production P_t: $Supply_t = P_t$
  - if `store` is selected, 10% of production P_t is stored (and vented if maximum capacity is reached) and 90% is supplied to the customer: $Supply_t = 0.9 \cdot P_t$
  - if `unstore` is selected, we supply all the production P_t and 10% of current storage: $Supply_t = P_t + 0.1 \cdot storage_t$

This context is overall illustrated in figure 6.1.

The objective of the supplier is to match the supply $Supply_t$ to the demand Y_t of our customer, unknown at the time of the decision of $\theta_t = (\theta_t^{prod}, \theta_t^{store})$, specifically, we consider the following

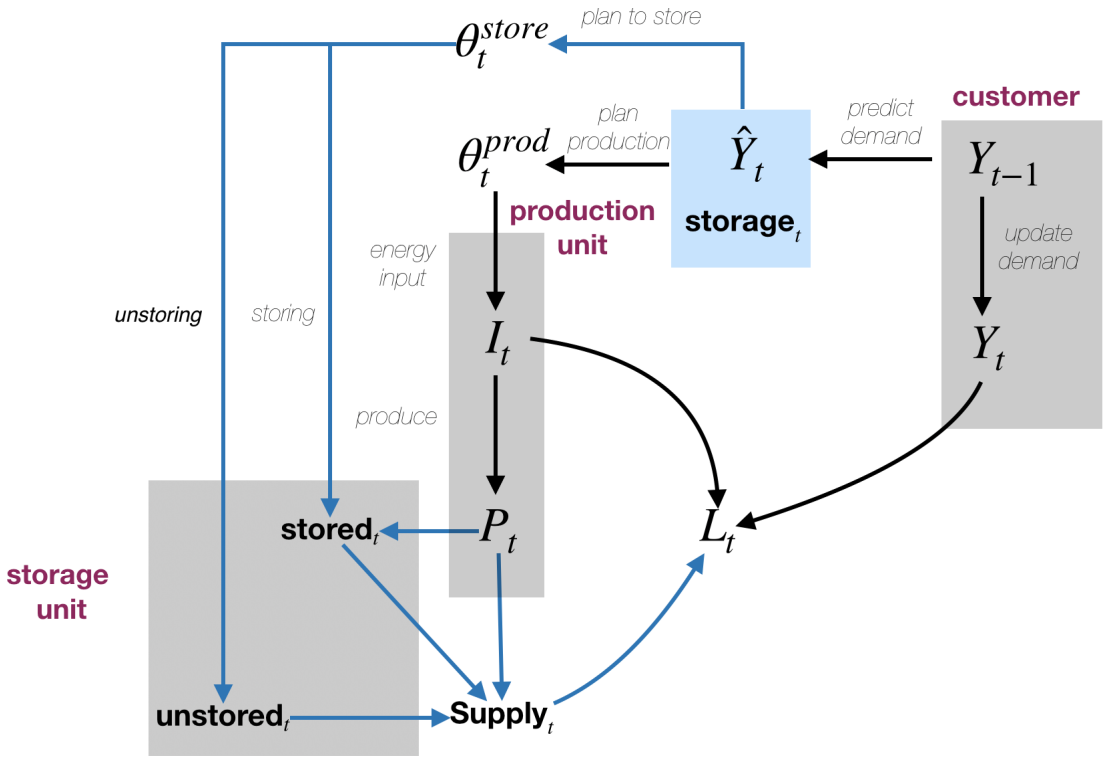

Figure 6.1: In comparison with the initial forecast-based optimization problem introduced in part I, we added a storage unit to the production facility. At each time step, we plan a production level and whether we add / reduce storage or leave it be. To facilitate comparison with the previous case the lines added are blue colored and we tried to seperate the entities with grey areas (customer side, production facility, storage facility) and added a purple label to each area.

loss function:

$$\begin{aligned}\mathbf{L}''(\theta, y, supply) = d \cdot I(\theta^{prod}) & \qquad \text{Input cost} \\ + e \cdot \max\{y - supply; 0\} & \qquad \text{Unmet demand cost} \\ - f \cdot \min\{y; supply\} & \qquad \text{Revenue}\end{aligned} \tag{6.1.1}$$

for the demand y, supply $supply$ and control parameters θ. Note that in our case the amount supplied directly depends on the current storage level and the couple of parameters θ.

As we only know demands up to time $t-1$ included, hence we will consider a decision made based on prior day demand Y_{t-1} and storage level at beginning of time t: $storage_t$. The storage unit should allow us to handle high-demand levels, higher than production capacity for a day, and potentially low-demand days, allowing to lower production and simply unstore product.

6.1.3 Two approaches

Let $\gamma \in (0, 1)$ denote a discount factor, a parameter left to the user to be chosen.

Forecast demand, then plan production

We consider the classic optimization problem, which assumes that demand at time t, written Y_t is known, and chooses how to best operate the industry (production level and storage decision) in order to maximize overall returns in the long-term:

$$\pi_1^*((y, s)) = \arg\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=1}^{+\infty} \gamma^{k-1} \mathbf{L}''(\theta_k, Y_k, Supply_k) | Y_1 = y, storage_1 = s\right] \tag{6.1.2 (Model 1)}$$

later used in conjonction with a prediction function of Y_t based on its antecedent Y_{t-1} :

$$\theta_t^* \triangleq \pi_1^*((\hat{f}(y_{t-1}), storage_t)). \tag{6.1.3}$$

This model will be referenced afterwards as model 1 and will be tested in three cases:

1. perfect prediction (i.e. knowledge) of Y_t,
2. an agnostic prediction based on Y_{t-1} and
3. a task-driven prediction based on Y_{t-1}.

To simplify the experiments we will focus on the class of linear models for the prediction functions.

Integrate demand uncertainty in production plan

We will test a second model which takes directly in input the previous demand Y_{t-1}, rather than Y_t, later substituted by its prediction. This model is defined as follows:

$$\pi_2^*((y, s)) = \arg\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=1}^{+\infty} \gamma^{k-1} \mathbf{L}''(\theta_k, Y_k, Supply_k) | Y_0 = y, storage_1 = s\right]. \tag{6.1.4 (Model 2)}$$

and can be directly used to select control parameters:

$$\theta_t^* \triangleq \pi_2^*((y_{t-1}, storage_t)). \tag{6.1.5}$$

This approach, referenced as model 2 afterwards, consists in changing directly the optimization problem rather than treating it as black-box and optimizing the input that is the forecast. Putting optimization issues aside, the minimum reached by model 2 should be lower than the minimum reached by model 1.

6.1.4 Process algorithm

The complete process described above and in figure 6.1 is unfolded in algorithm 7.

Algorithm 7: Process considered in this application.

```
   /* choose between model 1 and model 2 for control parameters          */
 1 choose control approach using_forecast ∈ {false, true}
   /* iterating over T time steps                                        */
 2 for t = 1, ..., T do
       /* select control parameters                                      */
 3     if using_forecast then
 4         Ŷ_t = f̂(Y_{t-1})
 5         θ_t = π*_1((Ŷ_t, storage_t))
 6     else
 7         θ_t = π*((Ŷ_{t-1}, storage_t))
 8     end
       /* produce, (un)store and supply                                  */
 9     P_t = P(I(θ_t^prod), θ_t^prod)
10     if θ_t^store == store then
11         Supply_t = 0.9 · P_t
12         storage_{t+1} = min{storage_t + 0.1 · P_t, MAX_STORAGE}
13     else if θ_t^store == unstore then
14         Supply_t = P_t + 0.1 · storage_t
15         storage_{t+1} = 0.9 · storage_t
16     else
17         Supply_t = P_t
18         storage_{t+1} = storage_t
19     end
       /* observe loss                                                   */
20     L_t = L''(θ_t, Y_t, Supply_t)
21 end
22 return (storage_t, θ_t, Supply_t, Y_t, L_t)_{t=1}^T
```

6.1.5 Demand scenario

We chose a demand scenario such that a storage unit would be justified i.e. on some days, the demand exceeds daily production capacity and therefore would require us to anticipate and store as much product as possible. On the other hand, the scenario allows for some low days also, such that we can actually produce in excess and store the product. Specifically, we considered an AR(1) demand model with auto-correlation 0.3 and bounded within [60 ; 400]. We considered a storage of size 200 units, whilst we can produce within range [90 ; 360]. For the loss function, similar

constants were used.

6.2 Result

To learn π_1^* and π_2^* we generated 5000 state-action-state-reward observations (in a random uniform manner for a good coverage) and used a neural network as function approximator following the generic Q-learning scheme, running it through 100 iterations, with $\gamma = 0.95$. The network is a fully-connected feed-forward network of five layers of 30 neurons each, taking as input the state and action (dimension 4: (storage, demand), (production decision, storage decision)) and outputing the Q-value estimate (dimension 1). We start the learning with a learning rate of 1e-4, optimizer rmsprop, 500 epochs maximum, and check for validation loss for early stopping (half of the data is left for validation). To simplify the process, we considered that production levels where by quarter (0%, 25%, etc). The test set initial state was generated randomly and for 1000 time steps.

In table 6.1 we present the average losses observed for the different models. Evidently the model with the knowledge of the true demand is the best one and consists of our baseline. It appears that once again using a task-driven approach for prediction is much more advantageous than taking an agnostic prediction for a specified task. Here, we also tested model 2, which bypasses the forecast task and directly changes the optimization problem. Such approach manages slight gains from the initial model with task-driven prediction.

As means of comparison we added the case where no storage unit is actually available, which corresponds to the application presented in part I. The resolution approach was also done using a neural network, although it wasn't ultimately necessary. Reproducing similar conditions allow us here to observe that in cases where the demand is forecasted (model 1 with agnostic or task-driven prediction) the average loss is higher when the storage is used than when it is never used, although, when the demand is known, there is significant loss drop. This means that because the optimization program was not built initially for a forecasted demand, the fact that we mis-use it with respect to its training leads to poorer performance in the setting where more flexibility is provided to the system, which is quite an interesting observation.

Three distinct points are to be made:

- comparing model 1 with perfect prediction performance with and without storage unit, we can see what the storage unit flexibility allows to bring in terms of pure loss for long-term demand

- comparing model 2 performance with and without storage unit, we can see what the flexibility from the storage unit allows to bring regarding long-term demand but also in handling the uncertainty of the prediction

- the models using forecasts performed better when no storage was involved

In figures 6.2 and 6.3 we present the production level and storage decision recommended by model 1 and 2 respectively. We also added as illustration an example of data sequence observed following model 1 under perfect information in figure 6.4.

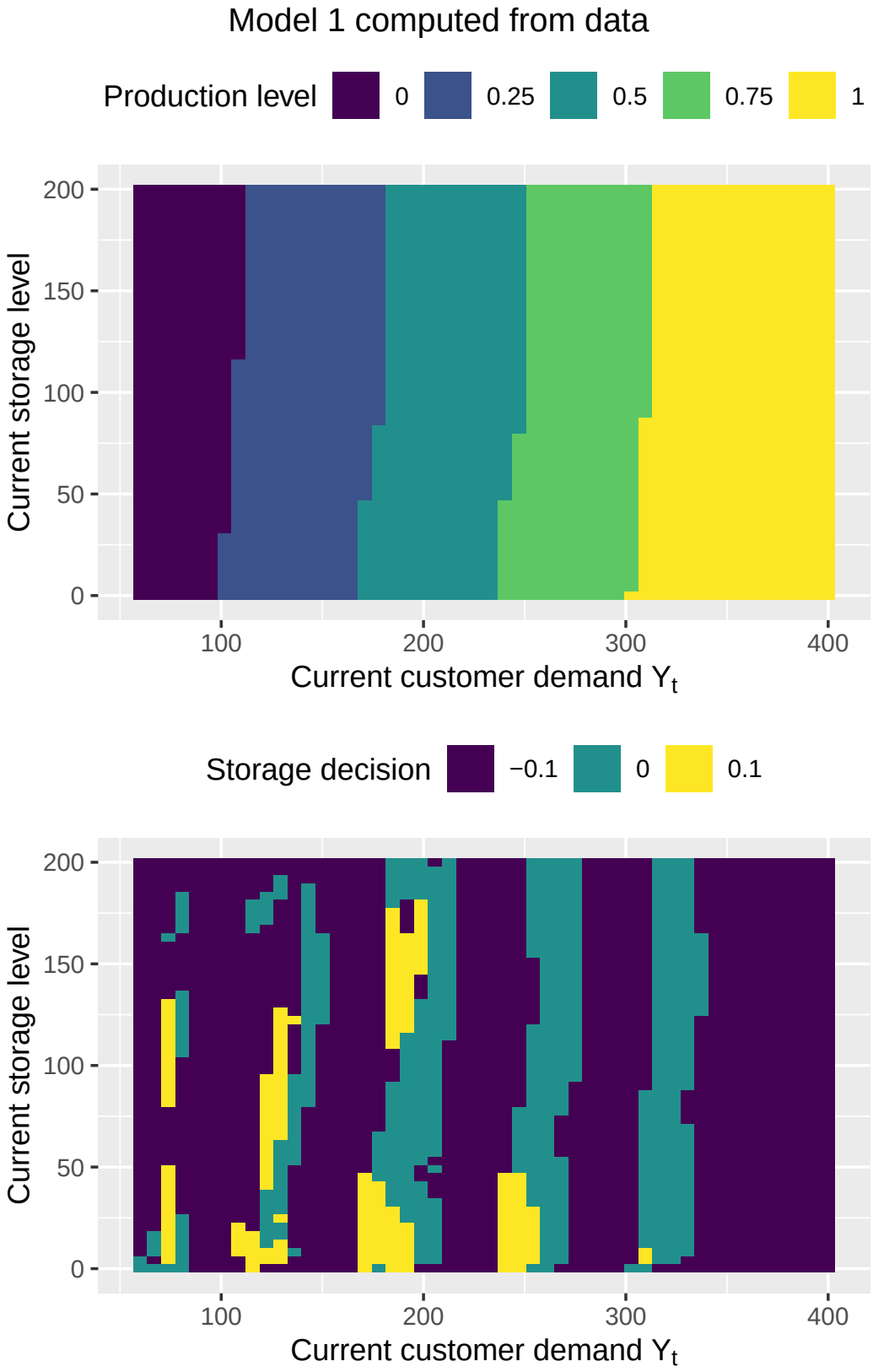

Figure 6.2: Model 1, providing the best production level and storage change couple considering current storage level and current customer demand Y_t. As one would expect, the higher the demand the higher the production, with the current storage level having a quite small effect. At each step where we choose to augment production we always store, then we don't then we unstore. The non-smoothness of this heatmap is due to the fact that we are using a neural network as function approximation which is smoothness-free. However, we should check for proper parameter convergence in order to finalize the approximation.

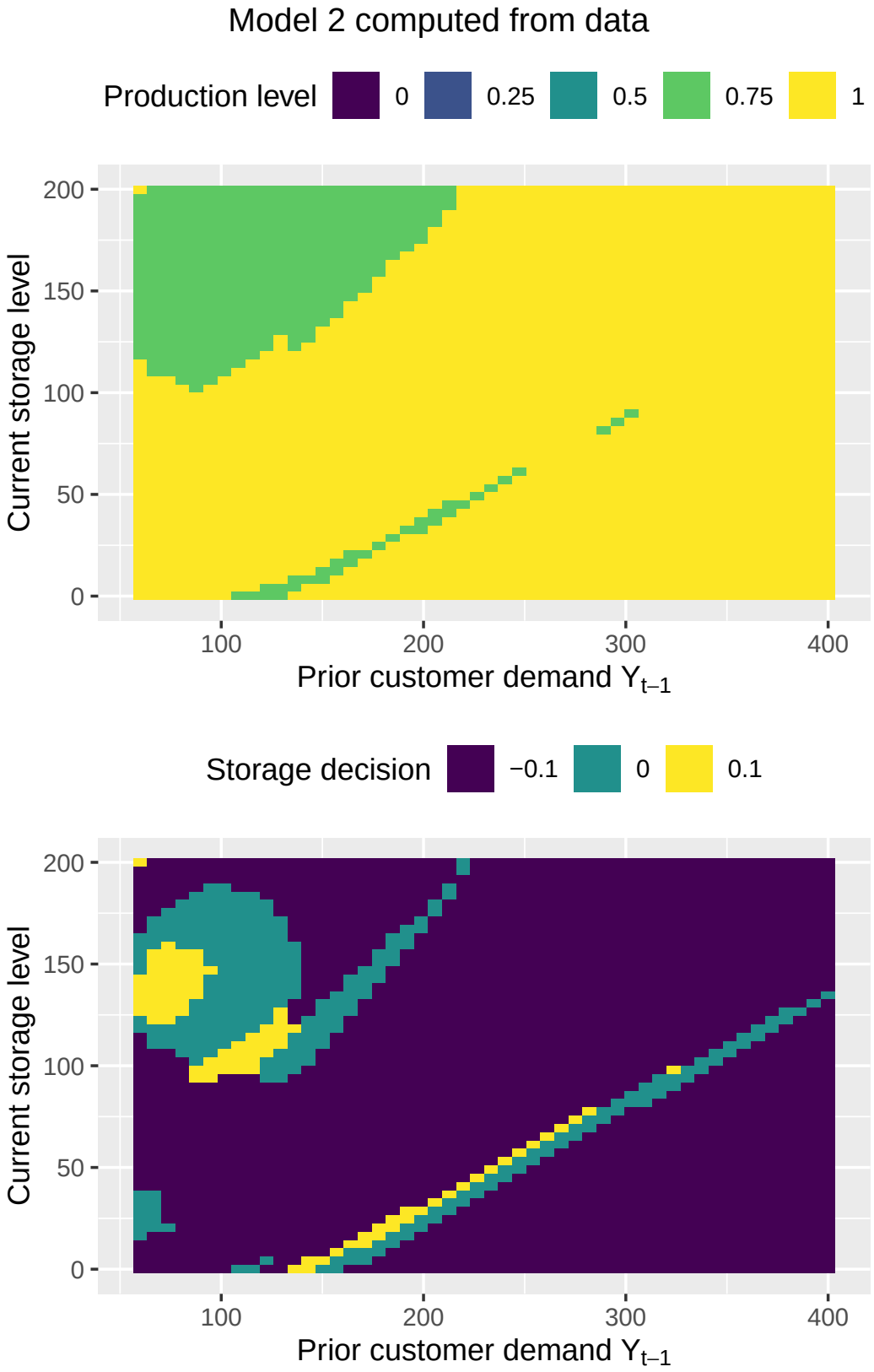

Figure 6.3: Model 2, providing the best production level and storage change couple considering current storage level and prior customer demand Y_{t-1}. Aside from small artefacts, it is common to produce at, at least, 75% of production capabilities. At the apparent frontier between producing 75% and 100% we see the same changes in storage decisions, yet much less smooth and relatively unclear (see the ghost-like shape). When demand is quite low and storage is not high enough, we exploit this situation by storing some product.

Model	Average loss on test set (StD)	
	with storage unit	without storage unit
Model 1 with agnostic prediction	2426.6	2116.6
Model 1 with task-driven prediction	150.6	133.6
Model 2	111.7	125.7
Model 1 with perfect prediction	-1266.5	-1183.0

Table 6.1: Average losses (and standard deviations) observed on test sets in models with sorage unit and without. The case where there is no storage unit is the one presented in the first part, although here the resolution approach is slightly different.

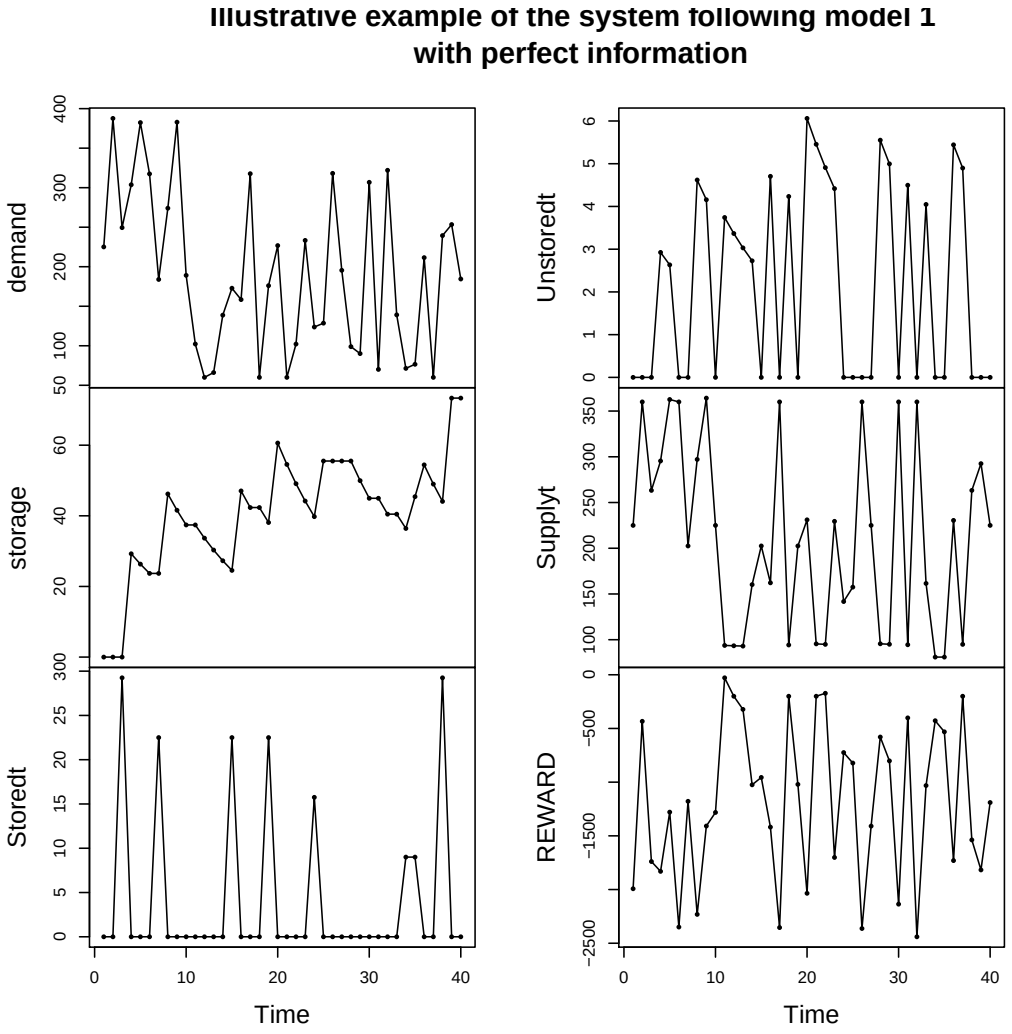

Figure 6.4: Illustration of the system controlled under Model 1 with perfect information. This allows us to see the dynamics of the storage unit imposed by the Model.

6.3 Conclusion

Similarly to the no-storage setting we considered in the first part, using a prediction based on a custom loss helps bridging the gap between the agnostic prediction performance and the perfect prediction loss. Because we added a storage and therefore a temporal side to the problem, we considered also not using a prediction module, but directly focusing on the joint storage and production decision problem. This led to even better than a simple task-driven approach. However, one should not forget that in practice, decision problems are already so complicated that building them based on initial features might be complicated, especially when target dynamics are much more complex e.g. large time dependencies, seasonality.

Chapter 7

Meal-based insulin-dosage recommendation for type I diabetes patients †

Summary

In the past two decades, Reinforcement Learning (RL) approaches have been given a lot of attention in many fields, often requiring substantial amounts of data. In [Kovatchev et al., 2009], the authors developed the T1DM simulator of the gluco-insulin interaction for diabetes management which was later approved by the FDA (Food and Drug Administration) as a substitute for preclinical trials (see The Epsilon Group [2017] for details on the development timeline and general information).

In this study, we propose a model-free approach to leverage a pertinent bolus advisor i.e. one which in the long-term performs well with regards to some reward function dependent on blood glycemia. We find smooth but more complex rules than the bolus advisor, more pertinent to a given meal scenario. Such result points to the idea that one could not only use this tool for insulin dose recommendation to a patient, but also to nutritional and potential physical activity program recommendation for an ideal blood glucose profile and overall positive health.

7.1 Context

Diabetes is a major disease in which the patient require insulin injection with adapted dosis via an insulin pump or injection pens. In this study, we challenge the classical dosis rule for patients of type I diabetes who do not have access to continuous monitoring. This corresponds to a patient producing low (if any) insulin, which makes blood glucose regulation through external insulin injection absolutely mandatory, in contrast with type II diabetes which is related to the progressive degradation of pancreatic activity, often related to nutritional issues. Note that diabetes of type I appears at a much younger age than diabetes of type II.

For those patients, the control of the blood glucose (BG) level is done with two type of injections: basal and bolus insulin. Basal insulin corresponds to a base level of external long-acting insulin. This base level substitutes the deficient pancreatical insulin production. The second type of injection, the bolus, are faster acting injections, made in order to help regulating carbohydrates (CHO) ingestion during meals.

The bolus injections are often taken around 10 to 30 minutes, prior to meal consumption. The amount of insulin should depend on the current blood glucose BG and the level of carbohydrates CHO that is going to be ingested. The goal is to maintain the blood glucose in the normoglycemic range [70 mg/dL; 180 mg/dL].

The most classical formula is given by the following piecewise linear formula : for a given measure of blood glucose BGand an intake of carbohydrates CHO, one should take :

$$\textbf{bolus}(\mathrm{CHO}, \mathrm{BG}) = \frac{\mathrm{CHO}}{\mathrm{CIR}} + \frac{\max(\mathrm{BG} - \mathrm{BG}_{target}, 0)}{\mathrm{CF}},$$

where the carbohydrate to insulin ratio (CIR) and the correction factor (CF) are specific to each individual, can be evaluated by medical tests and usually are fluctuant during the day.

Such a rule makes sense, we compensate meal glycemic intake as well as excess of current blood glucose, and is widely used in practice. However nothing is to attest for its performance in a sequential decision-making problem : what happens when we follow this rule several days and weeks ? Can't we find a more specialized rule ? Around the artificial pancreas topic, there has been multiple studies relying on proportional integrative derivative (PID) methods, model predictive control (MPC), Optimal control and Fuzzy techniques [Bothe et al., 2013]. Reinforcement Learning (RL) approaches progressively appeared in the early 2000s and more so in the following decade, with different use cases.

Authors of Ngo et al. [2018] for example investigated through RL methods the optimal insulin injection policy i.e. how to best deliver insulin to the body in order to have the best possible impact on glycemia. This was based on Insulin-Glucose kinetics model and gave insight into how to design a drug.

In Javad et al. [2019], authors apply Q-Learning algorithm on a cohort, aiming at finding an appropriate range of insulin dose to prescribe. They aimed at personalizing the treatment by representing each patient through some relevant information : Past Glycated Hemoglobin, BMI, Activity level, Alcohol usage, all discretized. Reward is then expressed depending on how Glycated Hemoglobin Evolved. One may note that the personalization is of course limited to the state definition. Also, such studies are difficult to conduct for any researcher, since they are conducted on real patient data and not simulated ones.

Based on a complex diabetes simulator, the authors of Daskalaki et al. [2013] proposed a double

Actor-Critic algorithm in the case of continuous glucose monitoring. They controlled and updated simultaneously the basal rate and the carbohydrate to insulin ratio, relying on the standard bolus advisor to prescribe bolus insulin. A more recent study from Sun et al. [2018], based on the same simulator investigated the update of the carbohydrate to insulin ratio between meals rather than daily, as well as robustness of the policy to uncertainty regarding skipped meals, meal sizes and times.

Here we propose to use a non parametric setting to show that this specific shape can be challenged. Using the vocabulary of Reinforcement Learning, we show how to learn non parametric policies, the function giving insulin dosis based on blood glucose readings and auxiliary information, that outperforms the classical one for a given meal pattern.

7.2 Markov Decision Process, Simulation and Reward

Action, States and Markov Decision Process

Our goal is thus to define an action a, the insulin dosis INS, which is a continuous value, from a state s described by two elements: one discrete ID_meal specifiying the meal time (and its known content), here either 6am, 12am, 3pm or 8pm, and a continuous one BG specifying the blood glucose level observed, a positive value, ranging from 20 mg/dLto 600 mg/dL.

To model our setting, we thus rely on a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{R}, \mathbf{T})$ as defined in chapter 5, with the state being the couple (ID_meal, BG) and the action being INS. The overall process is represented in figure 7.1.

In a first approximation, we may assume that our state perfectly defines the world so that an action a starting from a state s always yield the same state s', or at least such that the law of the state s' depends only on s and a. Such a transition law is assumed to be given by nature or at least as simulator.

Last but not least, in order to optimize the actions, we need a measure of quality of the choice i.e. we should obtain an information after our action measuring the quality of our solution. We assume thus we are able to measure a reward r after the action that is large if we are compliant to the blood glucose level and small if we are not. Note that, in our setting, there is no natural reward so that we will rely on the one proposed in Kovatchev et al. [2000], presented in the next subsection.

7.2.1 Reward definition

We propose to rely on a reward measure taking into account the whole trajectory from a given meal to the next one.

More precisely, we will use a scoring function h mapping a blood glucose level in $\mathbb{R}^+$ to a score in $[0; 1]$ such that h increases for blood glucose levels from 0 to a centered value of reference, and then decreases. For any meal i, consider T_i its time of occurrence and as such define the reward as:

$$R_i = \sum_{T_i < t \leq T_{i+1}} h(\text{BG}_t). \tag{7.2.1}$$

Ideally, one would like to keep blood glycemia in a standard range such as [70 mg/dL; 180 mg/dL]. Yet, it is inevitable that glycemia varies and travels out of such bounds, hence the need to quantify how good or bad is blood glucose level, acting as proxy for the patient's health.

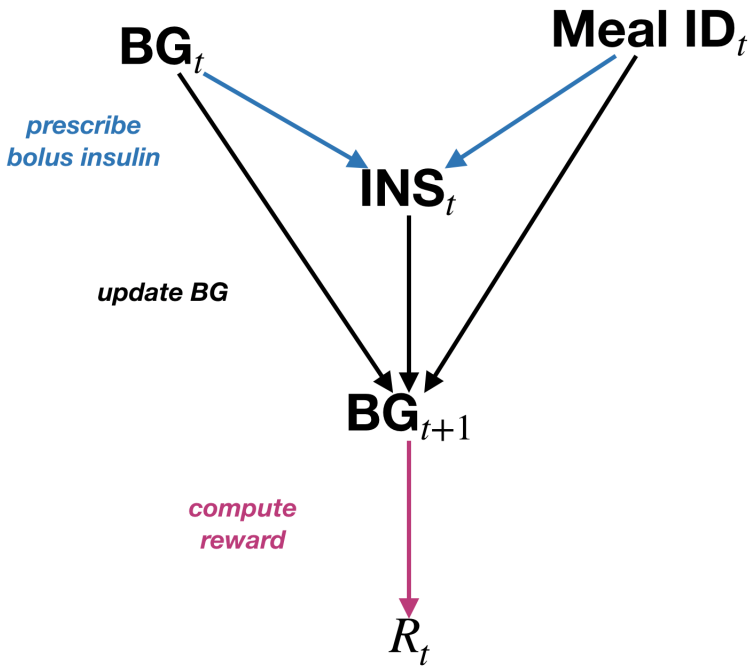

Figure 7.1: Illustration of the diabetes problem we are considering: at time t, given a blood glucose reading and a meal identifyer, $(BG_t,\ MealID_t)$, we propose a bolus insulin INS_t (blue lines: action). Then, we observe the next blood glucose state BG_{t+1} (black lines) and finally the reward associated to the BG level reached (purple lines).

Authors of [Kovatchev et al., 2000] proposed a risk function on the [0;100] scale based on symmetrization of the blood glucose levels and proper scaling proposed by [Kovatchev et al., 1997]. This reward function is illustrated in figure 7.2. Let us explain how it is built.

Consider a set of five (sorted) blood glucose reference levels

$$(\mathrm{BG}_{low}, \mathrm{BG}_{hypo\_threshold}, \mathrm{BG}_{center}, \mathrm{BG}_{hyper\_threshold}, \mathrm{BG}_{high}), \tag{7.2.2}$$

the authors [Kovatchev et al., 1997] define the following score function :

$$\forall \mathrm{BG} \in [\mathrm{BG}_{low}; \mathrm{BG}_{high}], \qquad h(\mathrm{BG}) = 1 - f(\mathrm{BG})^2 \tag{7.2.3}$$

with $f(\mathrm{BG}) \stackrel{\Delta}{=} \gamma(\log(\mathrm{BG})^\nu + \beta)$, where parameter set (γ, ν, β) is solution of the following set of equations :

$$\begin{cases} f(\mathrm{BG}_{center}) = 0 \\ -f(\mathrm{BG}_{hypo\_threshold}) = f(\mathrm{BG}_{hyper\_threshold}) \\ -f(\mathrm{BG}_{low}) = f(\mathrm{BG}_{high}) = 1 \end{cases} . \tag{7.2.4}$$

This parameterization implies that the risk function is at 0 (its minimal value) when blood glucose hits target BG_{center}, reaches its maximal value when it exceeds BG_{high} or plummets below BG_{low} and has equal value for $\mathrm{BG}_{hypo\_threshold}$ and $\mathrm{BG}_{hyper\_threshold}$. The authors Kovatchev et al. [2000] applied this log-transform for reference values (20, 70, 112.5, 180, 600) in mg/dLand established that low blood glucose index $(h(BG), BG < BG_{hypo\_threshold})$ is predictive of severe hypoglycemia episodes and high blood glucose index $(h(BG), BG > BG_{hyper\_threshold})$ is predic-

Figure 7.2: Reward function considered, taken from [Kovatchev et al., 2000] adapted to outer bounds (40 mg/dL, 350 mg/dL), rather than (20 mg/dL, 600 mg/dL), which where more appropriate for the patients we followed in the experiment section.

tive of glycosylated hemoglobin which is interesting for patient follow-up.

7.2.2 FDA Approved Simulator

In our study, we do not have access to the transition function or the reward, we only have access to a simulator of virtual diabetes patients that has been approved by the Federal Drug Administration in the USA as a substitute for preclinical trials. The T1DM simulator (2008 version, which we used here) is described in detail in Kovatchev et al. [2009]. It models the glucose and insulin dynamics taking into account

- for glucose : endogenous production, rate of appearance, utilization and renal extraction
- for insulin : rate of appearance from the subcutaneous tissue and insulin degradation.

It allows to obtain the evolution of the blood glucose for a given meal scenario and any insulin bolus policy with a sample rate of 20 observation per hour. It also provides good base values for Carbohydrate to Insulin Ratio, Correction Factor and Basal Rate.

With a 2,5 GHz Intel Core i7 processor and relying on the Python implementation of the simulator [Xie, 2018], it takes an hour to simulate a year of data which, sampled every three minutes comes to 175 200 samples. For the problem we consider, where we focus on bolus injection prior to meal times, this data will be reduced to a sequence of 1460 states and actions, as we will explain in the algorithms 8 and 9.

The simulator requires indeed a controller, which we call throughout this paper a policy i.e. a function mapping the state (meal and BG) to an action (Insulin level). Note that the behaviour policy of the simulator may be different from the one we are going to obtain by Reinforcement Learning.

In particular, it is known that we need to explore the state/action space to be able to derive a good policy. We have therefore used a behaviour policy which is quite different from the deterministic classical one. We will pick at random, quasi-uniformly the insulin level for a given state, within a range deduced from the deterministic classical one. More precisely, from patient records, one can find in the literature bounds for both scaling factors, as such we define, for any couple (CHO, BG) the maxima for insulin intake :

$$\bar{b} := \sup_{\substack{\mathrm{CIR}_{low}<\mathrm{CIR}<\mathrm{CIR}_{high} \\ \mathrm{CF}_{low}<\mathrm{CF}<\mathrm{CF}_{high} \\ \mathrm{BG}_{low}<\mathrm{BG}_{target}<\mathrm{BG}_{high}}} \frac{\mathrm{CHO}}{\mathrm{CIR}} + \frac{\max(\mathrm{BG}-\mathrm{BG}_{target},0)}{\mathrm{CF}}$$

with BG_{low} and BG_{high} bounds for target glycemic levels which should sensibly be within the normoglycemic range [70 mg/dL; 180 mg/dL]. Let $\underline{b}$ denote the infimum of the function, then we can consider prescribing insulin following a uniform distribution :

$$\pi^{behaviour}((\mathrm{CHO},\mathrm{BG})) = \mathrm{Uniform}([\underline{b},\bar{b}]).$$

Basal insulin will be set to the level indicated by the simulator's metadata on patients. In our application we will assume that the patient takes long-action insulin once or twice per day providing a quasi-constant insulin level.

This generation process is summarized in algorithm 8.

Algorithm 8: Data generation process

instanciate simulator Sim based on $meal\_scenario$
set basal rate $basal\_rate$ and behaviour policy $\pi^{behaviour} \in \Pi^{stochastic}$
sample first observations $O_1 = (\mathrm{BG}_1, \mathrm{CHO}_1)$
for $t \in \{1, \ldots, T\}$ **do**
    `/* pick insulin following policy` $\pi^{behaviour}$ `*/`
    if $\mathrm{CHO}_t > 0$ **then**
        $\mathrm{INS}_t = basal\_rate + \pi^{behaviour}((\mathrm{CHO}_t, \mathrm{BG}_t))$
    else
        $\mathrm{INS}_t = basal\_rate$
    end
    `/* sample following observations */`
    $O_{t+1} = (\mathrm{BG}_{t+1}, \mathrm{CHO}_{t+1}) \sim Sim(O_t, \mathrm{INS}_t)$
end
return $(O_t, \mathrm{INS}_t)_{t=1}^{T}$

As presented in algorithm 9, we reshape the data to match it to our problem : injection of insulin prior to meals. We therefore extract information at meal times and apply a reward function based on blood glucose levels (hence not directly observable in real life) which will characterize how good or bad is the blood glucose profile on the continuous [0;1] segment.

Algorithm 9: Shaping the data by meal times

input generated dataset $(O_t, \mathrm{INS}_t)_{t=1}^{T}$
input meal scenario used $meal\_scenario$
pick reward function $\mathbf{R}$, mapping set of BG levels to $[0; 1]$
identify $(T_i)_{i=1}^{nb\_meals} = \{t \in \{1, \ldots, T\} : \mathrm{CHO}_t > 0\}$
for $i \in \{1, \ldots, nb\_meals - 1\}$ **do**
    identify meal index $\mathrm{ID\_meal}_i$ from $meal\_scenario$, time
    `/* record state */`
    $S_i = (\mathrm{ID\_meal}_i, \mathrm{BG}_{T_i}, \mathrm{CHO}_{T_i})$
    `/* record action taken */`
    $A_i = \mathrm{INS}_{T_i}$
    `/* compute reward */`
    $R_i = \mathbf{R}(\mathrm{BG}_{T_i+1}, \ldots, \mathrm{BG}_{T_{i+1}})$
end
return $(S_i, A_i, R_i)_{i=1}^{nb\_meals}$

7.2.3 Reinforcement Learning and Function Approximation

Optimization function We can now formalize our policy optimization goal. Let Π the set of deterministic policies i.e. functions which assign to a state $s \in \mathcal{S}$ a single action $a \in \mathcal{A}$. We consider the following planification (policy optimization) problem :

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R_t\right]$$

where discount factor $\gamma \in (0,1)$ is chosen by the user. We define, for any couple $(s,a) \in \mathcal{S} \times \mathcal{A}$, state-action value functions $Q_\pi(s,a) \triangleq \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R_t | S_0 = s, A_0 = a\right]$ which allow us to write

$$\pi^* = \arg \max_{\pi \in \Pi, a \in \mathcal{A}} Q_\pi(s,a).$$

Q-Learning, basis function approximation In the case where state and action spaces are discrete, the preceding optimization problem can be solved via a tabular approach. One can use for instance the well known Q-Learning algorithm in an off-policy setting i.e. where the behaviour policy is not the optimized policy.

Considering that state and action spaces are continuous and might not be easily discretized without considerable information loss, we propose thus to use a function approximation of the state-action value function :

$$Q_\pi(s,a) \approx Q_{\bar{\alpha}}(s,a) = \sum_{b=1}^{B} \alpha_b \phi_b(s,a)$$

where $\bar{\phi} = (\phi_b)_{b=1}^B$ is a set of functions mapping $\mathcal{S} \times \mathcal{A}$ to $[0;1]$. Such a linear approximation scheme is widely used in the RL literature, some details can be found in chapter 9 of Sutton and Barto [2018].

The corresponding policy is defined implicitely through the parameter $\bar{\alpha}$ using the best choice strategy

$$\pi_{\bar{\alpha}}(s) = \arg \max_{a \in \mathcal{A}} Q_{\bar{\alpha}}(s,a).$$

We rely on the pseudo-gradient Q-Learning algorithm 10 to optimize vector $\bar{\alpha}$.

Algorithm 10: Q-Learning with linear basis function approximation

initialize $\bar{\alpha}^{(frozen)} = \bar{\alpha}^{(1)} = \vec{0}$
input set of transitions $\mathcal{D} = \{(S_t, A_t, R_t, S_{t+1}), t = 1, \ldots, T\}$
pick basis functions $\bar{\phi}$
pick learning rate $\eta, \eta > 0$, and discount factor $\gamma, \gamma \in (0;1)$
pick number of iterates before vector update $u, u \gg 2$
for $k \in \{1, \ldots, nb\_iteration\}$ **do**
  $t \sim \mathcal{U}\{1, \ldots, T\}$; `// sample transition set`
  $Y_k = R_t + \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\alpha}^{(\text{frozen})}}(S_{t+1}, a')$; `// compute target`
  $\bar{\alpha}^{(k+1)} \leftarrow \bar{\alpha}^{(k)} - 2\eta\bar{\phi}(S_t, A_t)\left(Q_{\bar{\alpha}^{(k)}}(S_t, A_t) - Y_k\right)$; `// Pseudo-Gradient Descent`
  If $k \mathbf{mod} u = 0$: $\bar{\alpha}^{(\text{frozen})} \leftarrow \bar{\alpha}^{(k+1)}$; `// update frozen vector`
end
return $\bar{\alpha}^{(K+1)}$

Gaussian Basis Function We propose to use the following linear decomposition of state-action value functions for our problem : given state $s = (\text{ID\_meal}, \text{BG})$ and action $a = \text{INS}$, we consider

$$Q_{\bar{\alpha}}(s,a) = \sum_{b}^{B} \sum_{b'=1}^{B} \alpha_{(b,b')}^{(\text{ID\_meal})} \phi_b^{\text{BG}}(\text{BG}) \phi_{b'}^{\text{INS}}(\text{INS})$$

which, by simple reindexing can be rewritten in the form presented previously. To distinguish between meals we assigned different set of parameters $\bar{\alpha}$, which is a standard way of distinguishing discrete values. Note that in our setting, the meal ID entirely controls the CHO level, hence it does not appear in the previous formula.

The basis functions ϕ_b^{BG} and ϕ_b^{INS} are radial basis functions, centered uniformly on their respective grids, as represented in figure 7.3. For the BG case (the INS is analogous), we consider, for any $b \in \{1, \dots, B\}$:

$$\forall x \in [\mathrm{BG}_{low}; \mathrm{BG}_{high}],\ \phi_b^{\mathrm{BG}}(x) = \exp\left\{-\frac{1}{2}\left(\frac{x - \mu_b^{\mathrm{BG}}}{h^{\mathrm{BG}}}\right)^2\right\}$$

where

$$\mu_b^{\mathrm{BG}} = \mathrm{BG}_{low} + b \cdot \frac{(\mathrm{BG}_{high} - \mathrm{BG}_{low})}{B}$$

and

$$(h^{\mathrm{BG}})^2 = \frac{(\mathrm{BG}_{high} - \mathrm{BG}_{low})/B}{2\log(1/(2p^{\mathrm{BG}}))}.$$

The set of expectation $(\mu_b^{\mathrm{BG}})_b$ is a uniformly-paced discrete grid of the range of BG values.

The variance term is chosen such that the area of ϕ_b^{BG} overlapping below μ_{b-1}^{BG} is around p^{BG}. Indeed: consider $X_b \sim \mathrm{Gaussian}(\mu_b, h^2)$ and $X_{b+1} \sim \mathrm{Gaussian}(\mu_{b+1}, h^2)$, then

$$\mathbb{P}(X_{b+1} < \mu_b) < \frac{1}{2}\exp\left\{-\frac{1}{2}\left(\frac{\mu_{b+1} - \mu_b}{h}\right)\right\}.$$

Bounding this probability by p set in $(0, 1)$ yields the expression of h presented above.

Basis function set for insulin is defined in the same way.

Note that if we where to take this parameter too low we would only estimate the state-action value function at the grid points $(\mu_b^{\mathrm{BG}}, \mu_{b'}^{\mathrm{INS}})_{(b,b')}$ whilst if too high we would oversmooth the state-action value function.

7.3 Numerical Experiments

For our experiments, we proceeded as follows :

- Step 0 : choose meal scenario of interest and virtual patient
- Step 1 : approximate optimal coefficients for the baseline bolus advisor via grid search (policy $\pi^{baseline}$)
- Step 2 : simulate a large amount of data following this meal scenario and exploring randomly the insulin intakes through meals with $\pi^{behaviour}$
- Step 3 : compute policy $\pi_{\bar{\alpha}^*}$ from simulated data via the Reinforcement Learning algorithm presented in the preceding section
- Step 4 : simulate new data following respectively $\pi^{baseline}$ and $\pi_{\bar{\alpha}^*}$, compare reward distribution and daily glycemic profiles and general glycemic indicators.

We chose the following meal scenario.

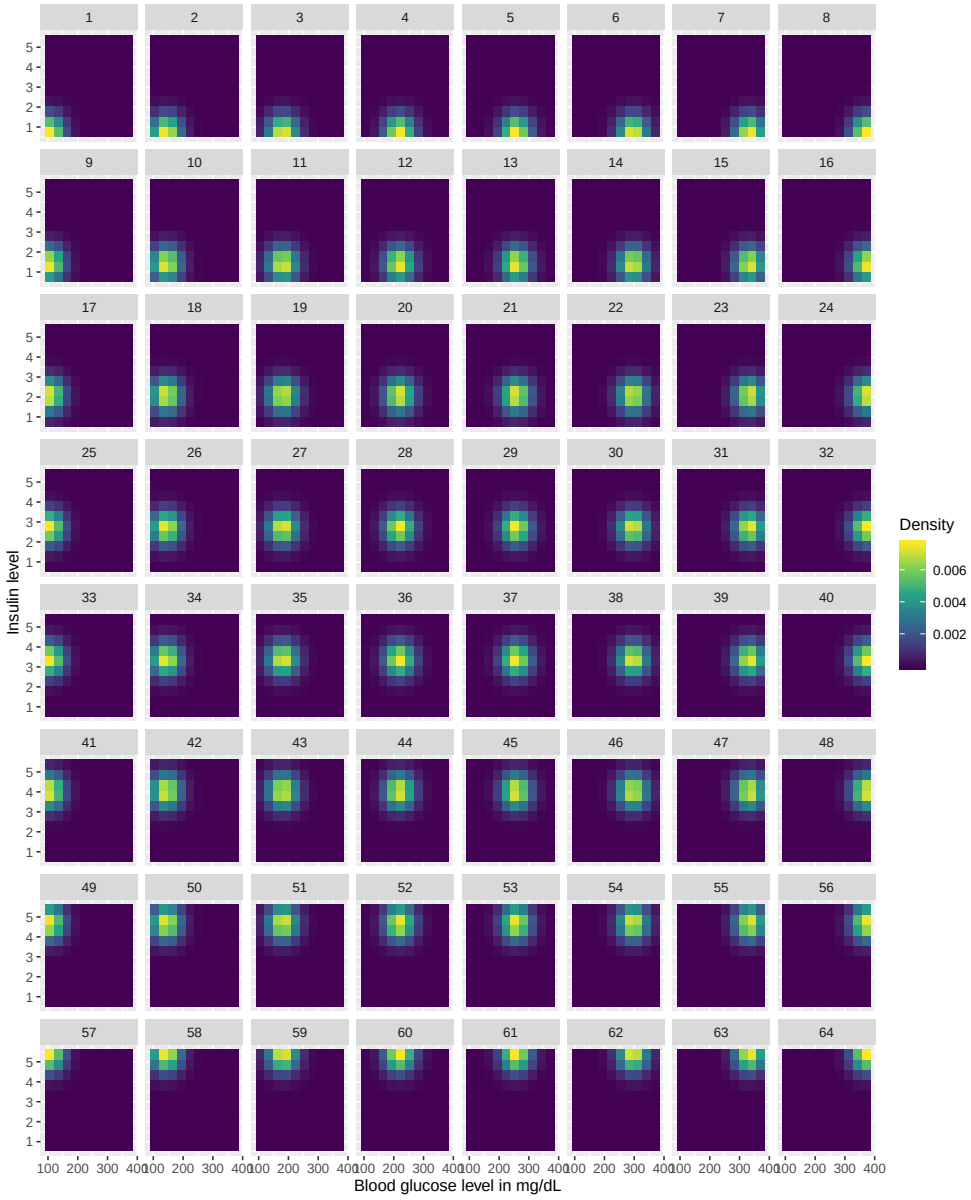

Figure 7.3: Illustration of the radial basis functions used to approximate the state-action value functions

Time of day	**Grams of CHO**
6 am	50
12 am	60
3 pm	15
8 pm	80

Simulation details The procedure above was applied to adults #001, #002 and #003 from the T1DM simulator. We simulated for each of them around 8 years of data which amounts to more than ten thousand transitions each. We based our grid search for $\pi^{baseline}$ from the following bounds : CIR $\in [3; 30]$, CF $\in [0.4; 2.8]$ and $\text{BG}_{target} \in [100 \text{ mg/dL}; 150 \text{ mg/dL}]$. For the Q-Learning algorithm, first we chose an 8 by 8 grid, and we set $p^{\text{BG}} = p^{\text{INS}} = 0.2$ which gave, from our qualitative evaluation, satisfying result. Then, we set the discount factor $\gamma = 0.75$ and the learning rate $\eta = 0.5$ for adults #001 and #002, we decreased it to 0.1 for the third patient because of divergence issues.

The policies found based on the initial data generated where then tested for the same virtual patients (separately, each adult has his/her policy) during 45 days, which is more than enough since the blood glucose enters a stationary state because meal scenario was set deterministic.

Result disposition In table 7.1 we reported general BG and reward indicators for each adult considered and for both policies. In figure 7.4 we present jointly the two policies for each patient, sorted by increasing index order. Because odd recommendations tend to arise in the policy computed for excessively large blood glucose levels, we jointly plotted the policy and the data which was used for its estimation procedure in figure 7.6 ; we show through this illustration that such behaviour is solely consequence of insufficient amounts of data in some regions of the space (BG, INS). In figure 7.5 we compare the average blood glucose behaviour between the two policies, for each patient, in order to see the concrete changes the optimized policy $\pi_{\bar{\alpha}^*}$ brings.

Analysis of table 7.1 It appears that for adults #001 and #003, our policy chose to prescribe overall less insulin since both patients are less in hypoglycemic levels following $\pi_{\bar{\alpha}^*}$ than $\pi^{baseline}$. For adult #002, the policy found chose to prescribe a bit more insulin to regulate slightly closer to the reference $\text{BG}_{center} = 112.5$ mg/dL. What must also be noted is that jointly the reward distribution was already quite excellent for adult #002 and we did not improve it according to the criterion and the same issue arises for adult #003 for whom hypoglycemia phases with $\pi^{baseline}$ where traded-off with hyperglycemia phases with our policy. All those observations can be confirmed from the averaged daily blood glucose profiles comparing the two policies in figure 7.5.

Policies found, figure 7.4 First, with our method we find smooth yet quite different policies than the baseline bolus advisor. Second, for a given blood glucose level, the recommended bolus does not necessarily increase with CHO amount ; we tend to give more insulin at lunch rather than at dinner, which impacts greatly the blood glucose profile. Typically the patient will be less at risk from nocturnal hypoglycemia which is a well-known issue in the endocrinology community. The policy we proposed properly took into consideration the meal scenario in its computations in order to look at future time steps and prevent complications. One may also note that the bolus advisors calibrated sometimes recommend decreasing amounts of insulin with increasing blood glucose level: a great example of that is the lunch recommendations for adults #001 and #002 : when blood glucose goes over 300 mg/dL the recommended insulin level drops. As is shown in figure 7.5 this is actually due to the scarcity of the data collected within this area so it should be considered as a numerical artefact.

Daily BG profile comparison, figure 7.5 Let us focus on Adult #001, whose data is represented in the upper graph. The baseline retained induces low blood glucose during the night

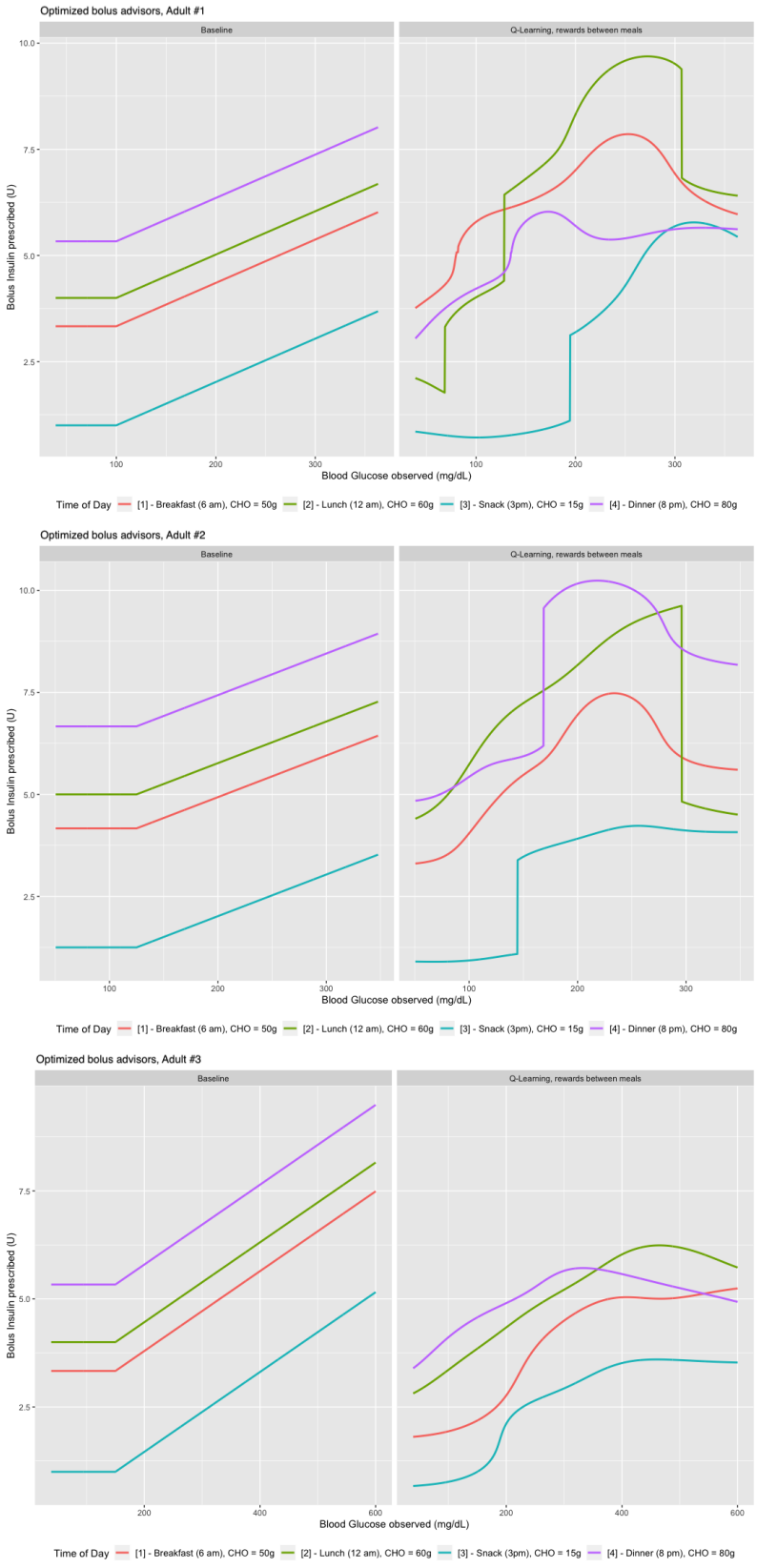

Figure 7.4: Bolus advisor $\pi^{baseline}$ (left) and $\pi_{\bar{\alpha}^*}$ found by Q-Learning with function approximation (right) for each virtual patient (sorted by increasing number). One can observe essentially two things from those graphs : first, it looks as if patient #001 and #002 are closer to each other than to patient #003 in terms of physiology otherwise we would not find the result we did. However the hope to get out a new generic rule seems difficult in this situation. Second, and this is what shows the algorithm took into account sequentiality of the meals and injections : the meal with most carbs does not necessarily require most insulin, which is a fundamental difference from the baseline advisor.

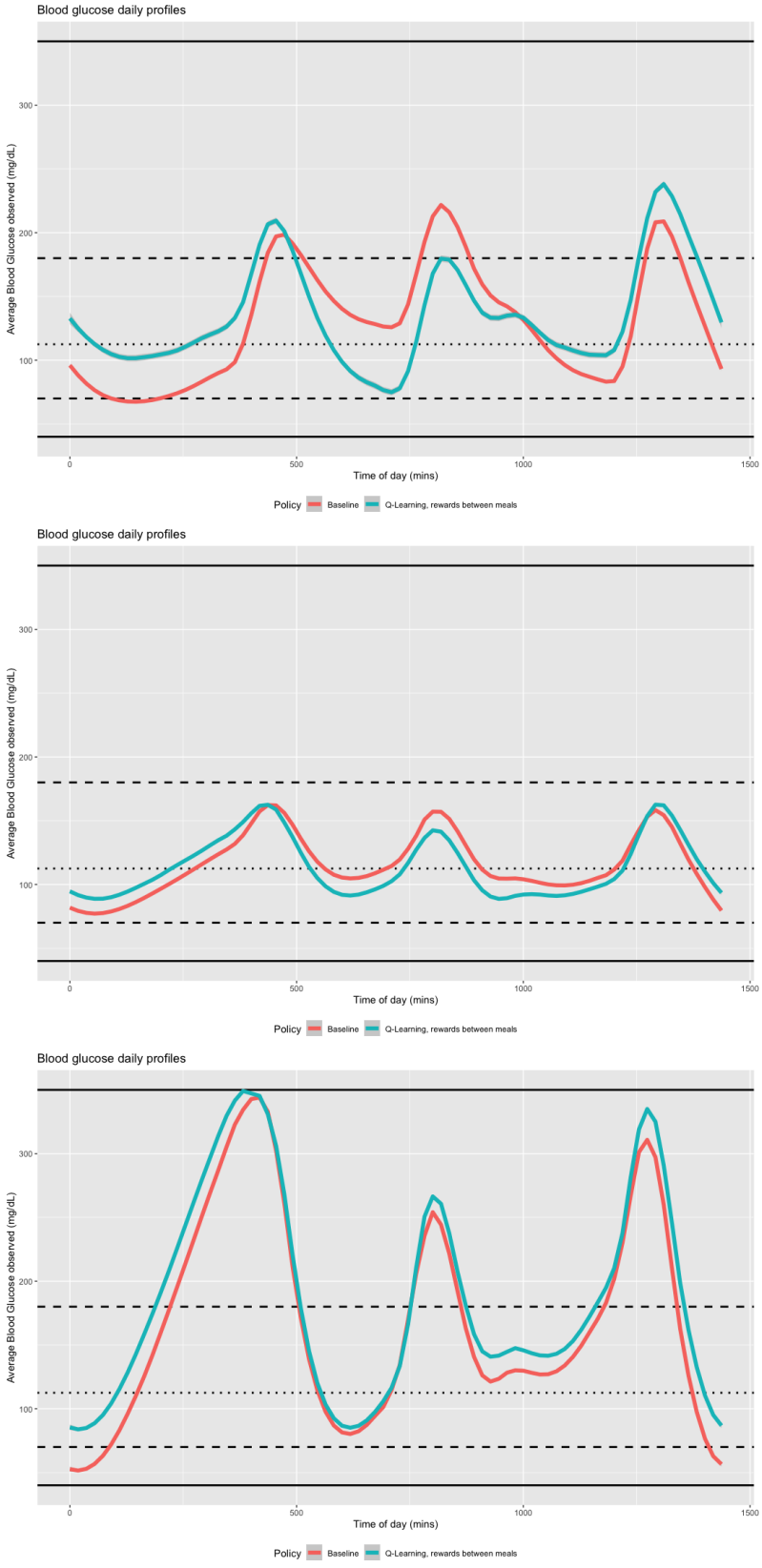

Figure 7.5: Average blood glucose profiles for the three virtual patients (sorted increasingly by number) for each policy applied, $\pi^{baseline}$ in red and $\pi_{\bar{\alpha}^*}$ in light blue. For all adults, the algorithm basically learned to reduce insulin intake at last meal in order to reduce nightly hypoglycemia at the cost of often minor hyperglycemia. The changes are minor in the case of Adult #002, but in Adult #003 and #001 it centers the BGlevels around target 112.5 mg/dL.

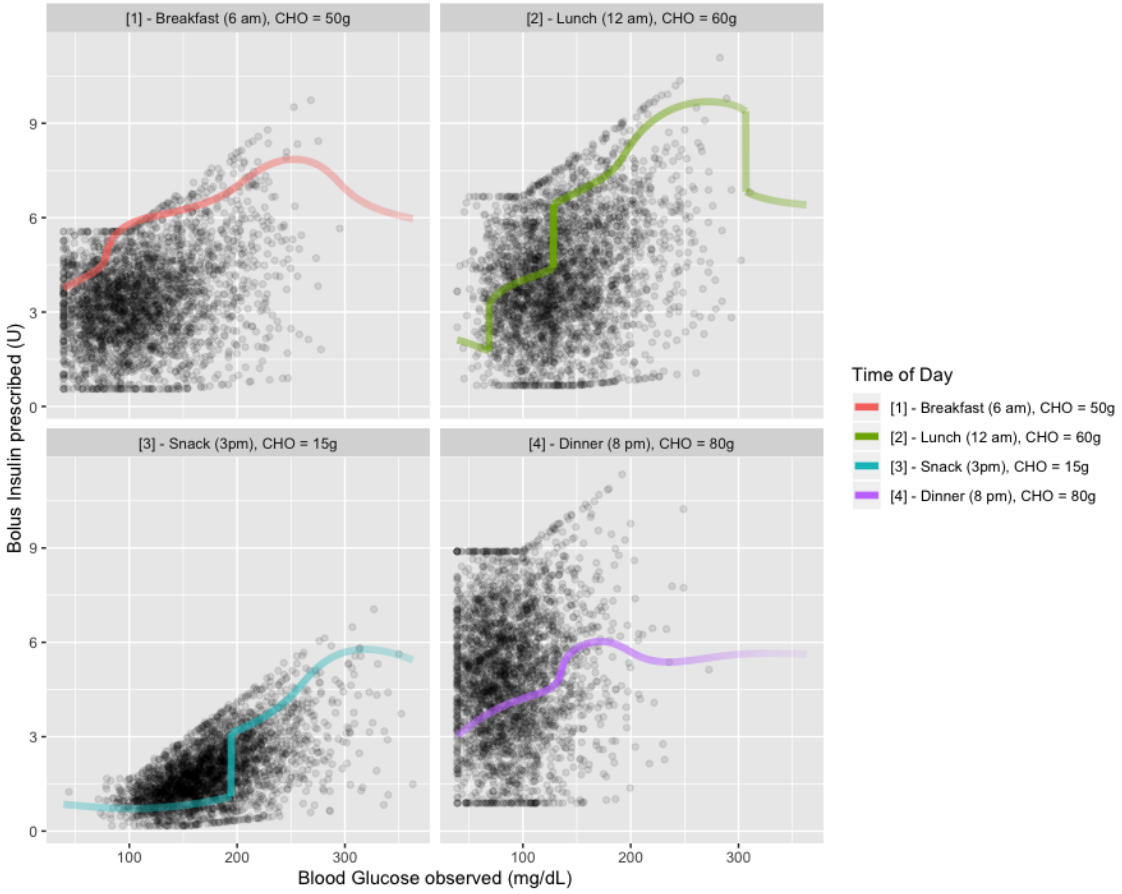

Figure 7.6: Focusing on adult #001, one can witness strange behaviours of the bolus advisor occurring on the extremities. Confronting it here with the simulated data points it is clear that such behaviours are due to insufficient amounts of data. In the context of simulation, since those regions are rarely visited one can leaves such policies unchanged. However, in real-life application, we should come up with a strategy to improve the quality of the policy in this region, or at least correct it, be it manually.

Indicator	Adult #001 π^{base}	Adult #001 $\pi_{\bar{\alpha}^*}$	Adult #002 π^{base}	Adult #002 $\pi_{\bar{\alpha}^*}$	Adult #003 π^{base}	Adult #003 $\pi_{\bar{\alpha}^*}$
Glycemia I						
$[40, 70)$	.07	.00	.00	.00	.08	.00
$[70, 112.5)$	.35	.37	.53	.58	.18	.20
$[112, 180)$	.39	.47	.47	.42	.35	.35
$[180, 350)$	.19	.16	.00	.00	.35	.39
$[350, 600]$	.00	.00	.00	.00	.04	.06
Glycemia II						
Low mean	68	NA	NA	NA	58	70
High mean	201	209	190	186	264	271
Reward						
Q1	.81	.88	.93	.95	.52	.48
Q2	.92	.96	.99	.98	.84	.88
Q3	.97	.99	1.00	.99	.97	.95
Mean	.88	.91	.96	.97	.72	.70

Table 7.1: Metrics useful to compare the baseline and computed policy. Low mean corresponds to the mean of glycemia level exclusively under the hypoglycemia threshold of 70 mg/dL and the high mean, analogously corresponds to the mean of glycemia level exclusively high the hyperglycemia threshold of 180 mg/dL. NA corresponds to non-applicable metrics e.g. if glycemia never drops below the bar of 70 mg/dL, the low mean indicator will be filled as NA.

(ranging in 70 mg/dL to 100 mg/dL) whilst it stays quite higher than 112.5 mg/dL from 6am to 3pm. The policy we found (in blue) eliminates night hypoglycemia, traded-off for a 2h period in mild hyperglycemia (<250 mg/dL) during dinner and a more smoothed out afternoon. For Adult #002, the two policies seem quite equivalent. For Adult #003, we trade-off the night hypoglycemic section by hyperglycemia on the three main meal times.

7.4 Optimizing jointly meal and insulin

Generating meal scenarios

1. Define an acceptable range for number of meals (2 to 6), meal times (between 6am and 10pm) as well as a constraint on total meal amount (in total 205 grams of CHO).
2. Pick a number of meals uniformly at random
3. Pick meal times uniformly at random (on hours)
4. Distribute meal amounts under total constraint uniformly at random (almost, rounding is applied)

For now, we assume that we don't know how to share information between different meal scenarios. We have seen that dynamic are linear, hence it would stand that for a given physiology and number of meals, we could extend the hypothesis by adding a smooth function over the meal time differences as well as meal amounts.

We found that the initial meal scenario we found rates third over the twenty random ones, which is somewhat reassuring for the people commionly following the three-meals a day rule. Now, although there is some issue with the scale apparently, it seems that one meal plan clearly stands far above others and that is the scenario reported in table 7.2.

Time of day	Grams of CHO
7 am	85
4 pm	52
9 pm	25
10 pm	43

Table 7.2: Best-performing meal plan

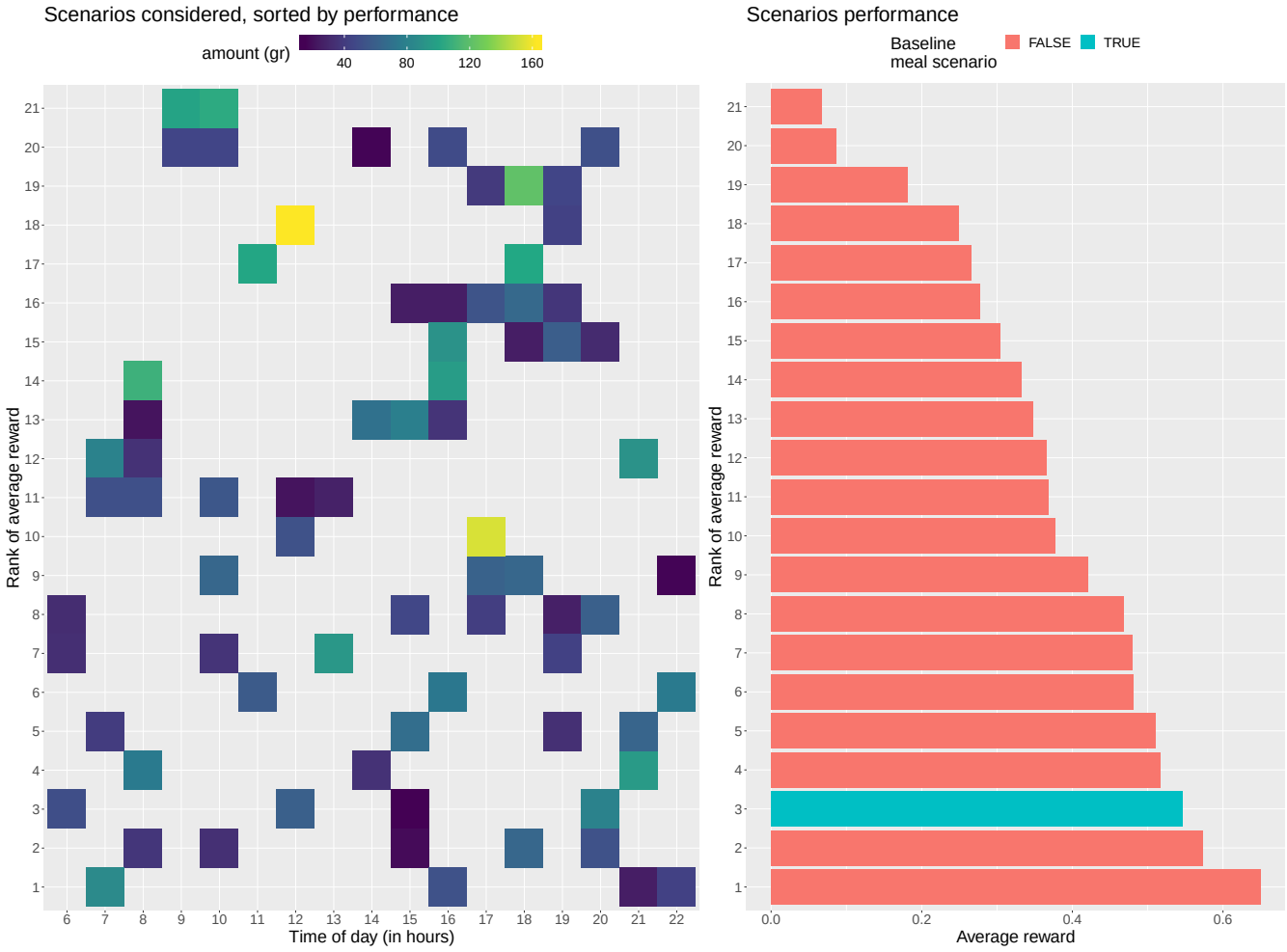

Figure 7.7: All meal scenarios considered ranked by overall performance in BG management (1 is best, 20 is worst). As a general observation, it appears that spreading out meals during the day overall helps in blood glucose management, as the first (best) nine meal plans tend to be spread meals throughout the whole day more than the eleven others.

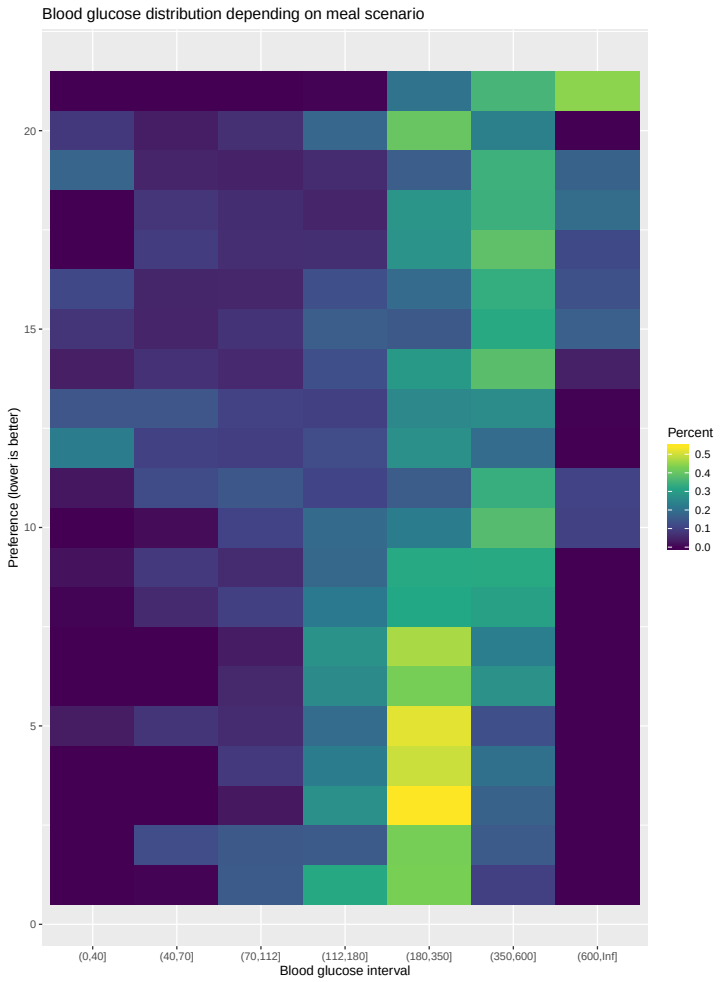

Figure 7.8: BG distribution by meal scenario. This graph gives generic information to compare the consequences of the different meal plans, for instance: meal plans 3-5 lead to BG essentially in interval (180 mg/dL,350 mg/dL], whilst the first two are more spread around smaller values. In fact the second meal plan is so spread it actually spends more time than the first one in interval (40 mg/dL, 70 mg/dL] which is not great.

Figure 7.9: Four best scenarios found. Quantitatively, the first meal plan really distinguishes itself from the others.

7.5 Discussion

7.5.1 Contributions

In most papers interested in Reinforcement learning for prescribing insulin injections, most of the research is done for patients equipped with insulin pumps and the goal is generally to estimate as quickly as possible the coefficients (CIR, CF) of the standard bolus advisor. In this paper, we challenge this heuristic by using a model-free approach from Reinforcement Learning. It turns out that we find policies which are (a) quite different from the baseline (b) are close-to parametric functions, which is so by construction (c) not necessarily intuitive in the sense they do not respect the following rule : insulin intake should increase with CHO amount. The policy for Adult #001 places for instance less insulin during dinner than on breakfast or lunch, although is the largest in terms of CHO. That way nocturnal hypoglycemia is avoided, which is very positive for the patient.

The framework we propose is also quite flexible to tailor a reward function to a patient e.g. to prevent hypoglycemia during a time window of the day, one can adjust the reward function for meal i previously introduced as follows :

$$R_i = \sum_{T_i < t \leq T_{i+1}} h(\mathrm{BG}_t, t).$$

Evidently the reward can also depend on observable auxiliary information e.g. physical activity which of course is highly relevant as we need to have proxys for patient's well-being which are as representative as possible.

Going further, we investigated a prescriptive approach where we tried different meal plans for a given virtual patient. We have tried, by a random, but properly constrained search, to find meal plans which allow easier blood glycemic control throughout the day. The best scenario seems to be taking a heavy breakfast of 85 grams of CHO at 7am, followed by a late lunch of 52 grams of CHO at 4pm and finally a dinner expanded on 9 and 10pm of 25 grams and 43 grams of CHO.

The code developed is available at `https://github.com/FredericLoge/T1DM_qlearning`.

7.5.2 Limitations & potential improvements

Simulator capabilities An important limitation lies with the simulator not taking into account activity information. Also, we should rely on the improved simulator from 2013 that requires to be in direct contact with Virginia Tech. Also, the Python implementation generates around 1500 observations in an hour, which is quite long. Hopefully the simulator from Virgina Tech works faster.

Using neural networks as function approximators By using neural networks, we would get surely much more precise results and we would be able to easily extend this approach in case other factors should be included such as external static information or activity information.

Uncertainties on future meal intakes We assumed that meal times and amounts were constant and strictly followed. This allowed us to see without adding noise from the randomness of nutrition if the baseline bolus advisor could be challenged which it clearly can. It would be very interesting to see how the policies we consider could adapt to the randomness in meal scenarios e.g. uncertainty in the amounts of CHO, meal skipped.

A prescriptive approach: planning meals and activities We have investigated different meal plans and focused on the blood glycemia control we could attain under those meal plans. We could go further by integrating activities and therefore deriving basic rules to avoid out-of-range periods.

BG uncertainty In the simulator measurement uncertainty is related to the installed machine. Yet in our target application we look at insulin pens, as such, we should investigate the measurement error from insulin pens to take it into consideration more neatly.

Reward personalization We relied on the log-transform reward because it is considered a standard in the field to analyze blood glucose levels precisely. Now, many other reward functions could be constructed.

- A two-sided radial basis functions with different scale parameter to equilibrate between hyperglycemia and hypoglycemia.

- As suggested in [Sun et al., 2019] and the authors prior work [Sun et al., 2018], one could consider :

$$h(BG_t) = c_{hypo} \max(\mathrm{BG}_{hypo} - \mathrm{BG}_t; 0) + c_{hyper} \max(\mathrm{BG}_{hyper} - \mathrm{BG}_t; 0).$$

- One could consider, in the spirit of the Control Variability Grid Analysis tool to take high and low quantiles of between meals blood glucose levels and build a reward function such as a scaled difference

- Another suggestion would be the maximum uninterrupted duration between meals where the blood glucose is within acceptable range.

Many possibilities will have to be considered.

Learning in real-life Our approach was from a discovery perspective, as we generated years of data with randomized policies in order to find what suited best a given physiology. This is completely infeasible in practice asis. There is a clear need to investigate how to setup a framework to find an appropriate policy from initial information, estimate required parameters as quickly as possible.

7.6 Conclusion

In [Kovatchev et al., 2009], the authors developed the T1DM simulator of the gluco-insulin interaction for diabetes management which was later approved by the FDA as a substitute for preclinical trials (see The Epsilon Group [2017] for details on the development timeline and general information).

The simulator takes into account meal times and amounts (in grams of carbohydrates) as well as an insulin controller, parameterized by the user. Submitted to the FDA in 2008, it was approved as a substitute for preclinical trials, which encouraged its use for PID, MPC and RL research of insulin-based glucose management.

Several studies of RL have been done on this simulator, most of which focus on the following bolus advisor :

$$\mathbf{bolus}(\mathrm{CHO}, \mathrm{BG}) = \frac{\mathrm{CHO}}{\mathrm{CIR}} + \frac{\max(\mathrm{BG} - \mathrm{BG}_{target}, 0)}{\mathrm{CF}},$$

where CIR is a carbohydrate-to-insulin ratio and CF is a correction factor. Both parameters are specific to each individual as they are dependent on physiology.

In this paper, we searched for a pertinent challenger to the baseline bolus advisor above with a model-free approach from Reinforcement Learning mixing Q-Learning with linear function approximation. We found smooth but more complex rules than the bolus advisor, more pertinent to a given meal scenario; in particular, the policy lowered the risk of nocturnal hypoglycemia.

If one could use his/her physiological constants to instanciate a simulator this would mean a lot from the perspective of both insulin injection, nutritional knowledge and potentially more if the simulator allows it, namely physical activity. In other words, one could not only use this tool for insulin dose recommendation to a patient, but also to nutritional and potential physical activity program recommendation for an ideal blood glucose profile and overall positive health.

Acknowledgments

We would like to thank our colleague at Air Liquide, Fouad Megherbi, MD, Medical Director, for his input on this topic and medical references he provided us with to enrich the study.

Communication

This topic was presented at the Data Science for Health (DSHealth 2020) workshop which is part of the KDD conference. This workshop focused essentially on research topics which brought insights related to Health topics.

Chapter 8

Optimizing decisions in run-in period †

Summary

In this chapter we consider a two-phased problem, modeling the installation and use of a device set up for a user, patient, customer. The first phase, relatively short, is the running-in period, where several milestones are clearly defined, with specific actions. The task tackled here consists in choosing appropriate actions so that, in the long-run, (a) the service provided is of good quality and (b) it is adopted by the user. Evidently, its adoption will depend in the ability to provide a personalized servicen hence the importance of such actions, especially at the beginning of service.

8.1 Generic problem

Because of confidentiality, we choose to speak of a quite general process, although we focused on a very specific problem for Air Liquide.

The generic problem is presented in figure 8.1. The process of interest is played through two periods: the first being a run-in period and the second being more stationary. The idea is that decisions made during the first period set the frame for the second one e.g. setting up a device, building a piece of software, meeting a customer.

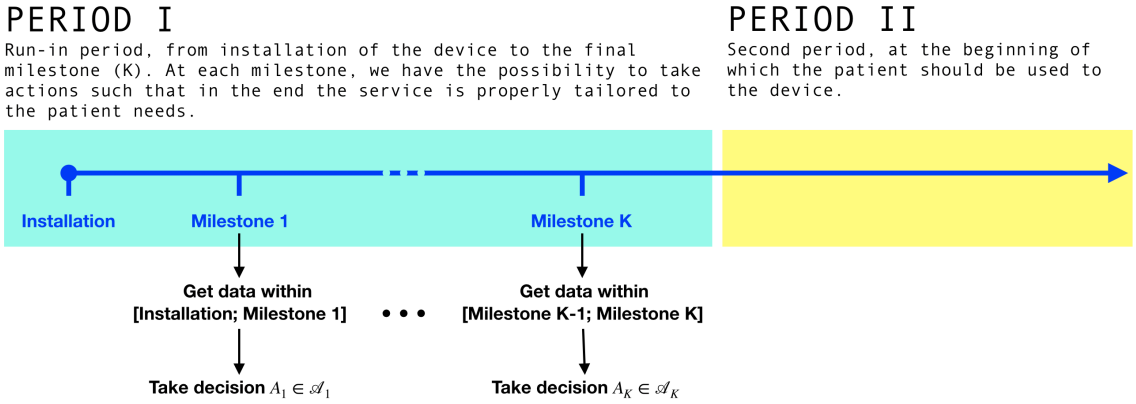

Figure 8.1: Schematic of the process: two periods compose this process, the first one being a run-in for the second, going from the installation of the device, through several milestones where important adjustments can be made before entering the stationary period where the service is supposedly then properly tailored to the patient.

We focus on the problem of setting up a device at a patient's home. Two quantities are measure during this process: observance and quality. Observance meaning how much the patient actually uses the device, a quantity bounded in [0; max_observance]. Quality meaning the quality of the service as a whole i.e. the device, its proper installation etc. Of course one wants to have a good quality for everybody and a top-notch service, but you can't help the fact that adjustments and personalization are required to actually provide a proper service. The one size fits all approach does not work. The idea is that such adjustments can be made at specific milestones, with an incurred cost and the mathematical objective is to balance how observant patients are and how much we invest in order to improve the quality of service i.e. finding policy

$$\pi^* \triangleq \arg\max_{\pi\in\Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{T} \mathbf{reward}(observance_t) - \mathbf{cost}(action_t) \right] \tag{8.1.1}$$

asssuming T is finite and the **reward** and **cost** functions map to a bounded interval of $\mathbb{R}$. To be specific, in the previous equation, the index t corresponds to a specific time unit of the problem, and milestones have specific times $(T_i, i = 0, \dots nb\_milestones)$, all in $\{0, \dots, T\}$, $T_0 = 0$.

Note that this of course one approach to characterize this balance between observance and action-taking and other metrics could be of interest. In our approach, we equivalently consider the following problem: finding policy

$$\pi^* \triangleq \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{i=1}^{nb\_milestones} \underbrace{-\mathbf{cost}(action_{T_i})}_{\text{cost of action at milestone } i} + \underbrace{\mathbf{reward}((observance_t, T_i \leq t < T_{i+1}))}_{\text{reward for the period following milestone } i} \right]. \tag{8.1.2}$$

Note that here we purposefully ignore the observance levels prior to the first action, since we assume we will treat patients fairly and so no sense to compare values prior to the first action taken.

The solution we propose is to model the system as a Markov Decision Process, running with timesteps as the milestones of the problem and which will include in the state information: the milestone, the past quality and the past observance. In the action we will have the action actually taken ; supposedly, at each milestone we have different types of actions possible.

The application of interest here is a textbook case of multi-step decision-making, as analyzed in Carpentier et al. [2015] and in episodic problems in Reinforcement Learning e.g. the cliff-walking problem and the Gambler's problem, see chapter 3.4 of Sutton and Barto [2018].

In order to solve either of those optimization problems, we will exploit the fact that the process is episodic and sequentially estimate the Q-values for the last milestone up to the first one. We will try both (1) discretizing the state space and (b) using a neural network for function approximation, which does not require feature engineering for the state space. Note that, of course, if the action space is continuous, it should also be discretized in case (1). In our setting we will assume a discrete action space, although it does not incur much on the process. We will try to also present dummy baselines for proper comparisons.

8.2 Instanciation

In order to present our approach, we have to add specifics to the problem: a simulator, determining the dynamics of the system, a clear definition of states variables and finally proper reward and cost functions.

The dynamics of the process are presented in algorithm 11 and the parameters in the algorithm are taken as expressed in table 8.1. In summary: the initial quality level and observance levels and drawn at random. Observance tends towards the point $quality \cdot \max\_observance$ with a speed defined by a parameter β and variations occur with additive gaussian noise of level σ. The quality level is stable throughout time, unless an action is done, which may improve or worsen it. The action effect is specified by function *change*, which is our case is defined such that

$$quality_{t+1} \sim \text{TruncatedGaussian}(mu = \mu(quality_t, action_t, t), sd = 0.01, min = 0.05, max = 0.95)$$

where

$$\mu(quality_t, action_t, t) = \begin{cases} quality_t - 0.1 & \text{if } t = T_1 \text{ and } action_t = 0 \\ quality_t - 0.05 & \text{if } t = T_2 \text{ and } action_t = 0 \\ quality_t & \text{if } t = T_3 \text{ and } action_t = 0 \\ \max\{quality_t + 0.2, 0.5\} & \text{if } t = T_1 \text{ and } action_t = 1 \\ \max\{quality_t + 0.1, 0.5\} & \text{if } t = T_2 \text{ and } action_t = 1 \\ \max\{quality_t + 0.05, 0.5\} & \text{if } t = T_3 \text{ and } action_t = 1 \end{cases}$$

such that, in average it is more interesting to take action than not, especially so at the first milestones.

We will therefore consider that our state at a given milestone time T_i is the set

$$\left(T_i, quality_{T_i}, \frac{1}{T_i - T_{i-1}} \sum_{T_{i-1} < t \leq T_i} observance_t \right),$$

the actions at each milestone or in $\{0, 1\}$ encoding respectively no action / action. The cost associated to an action is fixed to constant 50 and the reward is based on the average observance of the patient, which is our indicator of how well the device has been adopted and how much useful it has become in the patient's daily life. The device is supposed to be used each day, up-to 8 hours, hence the observance is bounded in [0, 8 hours].

8.3 Results

We considered two different approaches to solve our problem; first, discretizing the state space, with 5 bins for quality and average observance, based on data distributions, built irrespective of the milestone. Second, using a neural network for function approximation, with the following structure: four neurons as input (three for state, one for action), one neuron as output, and between those 6 layers of 15 neurons, with ReLU activation function and as final activation leakyReLu, to prevent problems due to unfortunate weight initialization. Optimizer rmsprop was used with learning rate 0.005 and a mean-squared error as loss function. To facilitate training we normalized the past average observance as well as the reward minus cost variable.

The training data generated for those policies consisted of 10000 simulations.

Three baseline policies were considered: the deterministic policies where at each milestone we pick action 0 or action 1 and the random policy where we choose action 0 or 1 with equal probability.

The test data for the five policies was taken on 1000 simulations, with an initial seed to encourage proper comparisons.

We compare policies in two ways: in table 8.2 we look at the proportion of cases overall where each method prescribes to take an action, to have an idea on how the two approaches of interest behave in average. Then, in figures 8.2, 8.3 and 8.4 we look at the policies computed for both approaches for milestones 1, 2 and 3 respectively. Specifically, we look at the evaluated difference

Algorithm 11: Process dynamics

input parameter β observance rate of change between time steps
input function *change* which to a given quality level and a given action assigns a new quality level stochastically
input policy π
$quality_0 \sim \text{Uniform}([0;1])$
$observance_0 \sim \text{Uniform}([0; \text{max\_observance}])$
$action_0 = 0$
for $t = 1, \ldots, T$ **do**
    /* update quality */
    if $t - 1 \in \{T_1, \ldots, T_{nb\_milestones}\}$ **then**
        $quality_t = change(quality_{t-1}, action_{t-1}, t-1)$
    else
        $quality_t = quality_{t-1}$
    end
    /* measure observance */
    $observance_t =$
     $observance_{t-1} + \beta(quality_t \text{max\_observance} - observance_{t-1}) + \text{Gaussian}(0, \sigma)$
    /* observe action */
    if $t \in \{T_1, \ldots, T_{nb\_milestones}\}$ **then**
        $i \in \{1, \ldots, nb\_milestones\} : T_i = t$
        $action_t = \pi\left(\left(t, quality_t, \frac{1}{T_i - T_{i-1}} \sum_{T_{i-1} < t' \leq T_i} observance_{t'}\right)\right)$
    else
        $action_t = 0$
    end
end
return $(quality_t, observance_t, action_t)_{t=0}^{T}$

Label	Parameter	Value
Time unit	-	day
Rate of change of observance	β	0.01
Standard deviation for next observance	σ	5
Maximum observance possible	max_observance	480 (8 hours)
Number of observations	T	270 days
Milestones	T_1, T_2, T_3	30, 60, 90 days
Action space, all milestones	$\mathcal{A}$	$\{0, 1\}$
Cost of action A	$\mathbf{cost}(A)$	$50 \cdot \mathbb{1}\{A = 1\}$

Table 8.1: Parameters and details

	Milestone 1	Milestone 2	Milestone 3
Random, $A \sim$ Bernoulli(0.5)	47.5	49.1	48.7
Deterministic, $A = 0$	0	0	0
Deterministic, $A = 1$	100	100	100
Discretized	100	46.1	0.0
Q-Network	95.2	51.8	0.0

Table 8.2: Proportion of cases where an action was taken, by step in the process

of taking versus not taking an action:

$$\max_{\pi} Q_{\pi}(s, a = 1) - \max_{\pi} Q_{\pi}(s, a = 0)$$

such that we see retain the information of how much estimated difference there is to take or not the action at the current milestone. Hence, if the estimate is positive, it seems advantageous to take the action, whereas if it is negative it seems advantageous to take no action at this particular milestone. The upper graphs are the policies we have based on the discretized state space and the lower graphs are the policies based on the neural network. Note that the differences in scale between the two graphs are due to the normalizations operated in the neural network case. Note also that because of data distribution, inherent to the system dynamics, some parts of the state space are rarely explored. As such, some couples of quality and past average observance sometimes have no estimates presented. We added data hexograms of the training data for each milestone to emphasize this point, see figure 8.5. Note that for the neural network method, it can very well provide predictions outside the bounds of the training sample space. As such, we took it upon ourselves to get rid in our graphs of areas where very little training data is available (15 observations at least per tile plotted).

Looking at the proportions of actions taken it appears that both policies decided to act almost always at the first step, half the time the second step and never the third step. Now, using the neural network for evaluating our decisions, we tend to act slightly more at step 2 than 1 (6% difference). Looking at the decision surfaces, it appears that, for this particular problem instance, a simple linear decision boundary could be used. In a more complex setting however, one might see the point of being able to represent decision boundaries at each milestone.

Now, let us look at the performances on the set of the two policies, and compare them to basic baselines. When no action is taken ever, the average performance is 655, whilst it goes to 740 if we always take action. Using the discretized approach, we reach an average performance of 781 and using the neural network approach, we reach 798, which is quite an improvement in relative terms.

Method	Mean (± std)	Min.	1st Qu.	Median	3rd Qu.	Max.
Random, $A \sim$ Bernoulli(0.5)	704 (± 9)	47	509	702	891	1343
Deterministic, $A = 0$	655 (± 9)	30	450	648	861	1352
Deterministic, $A = 1$	740 (± 8)	175	570	734	920	1249
Discretized	781 (± 8)	227	586	775	964	1358
Q-Network	798 (± 8)	161	616	795	985	1380

Table 8.3: Performances of methods, including baselines, on test set

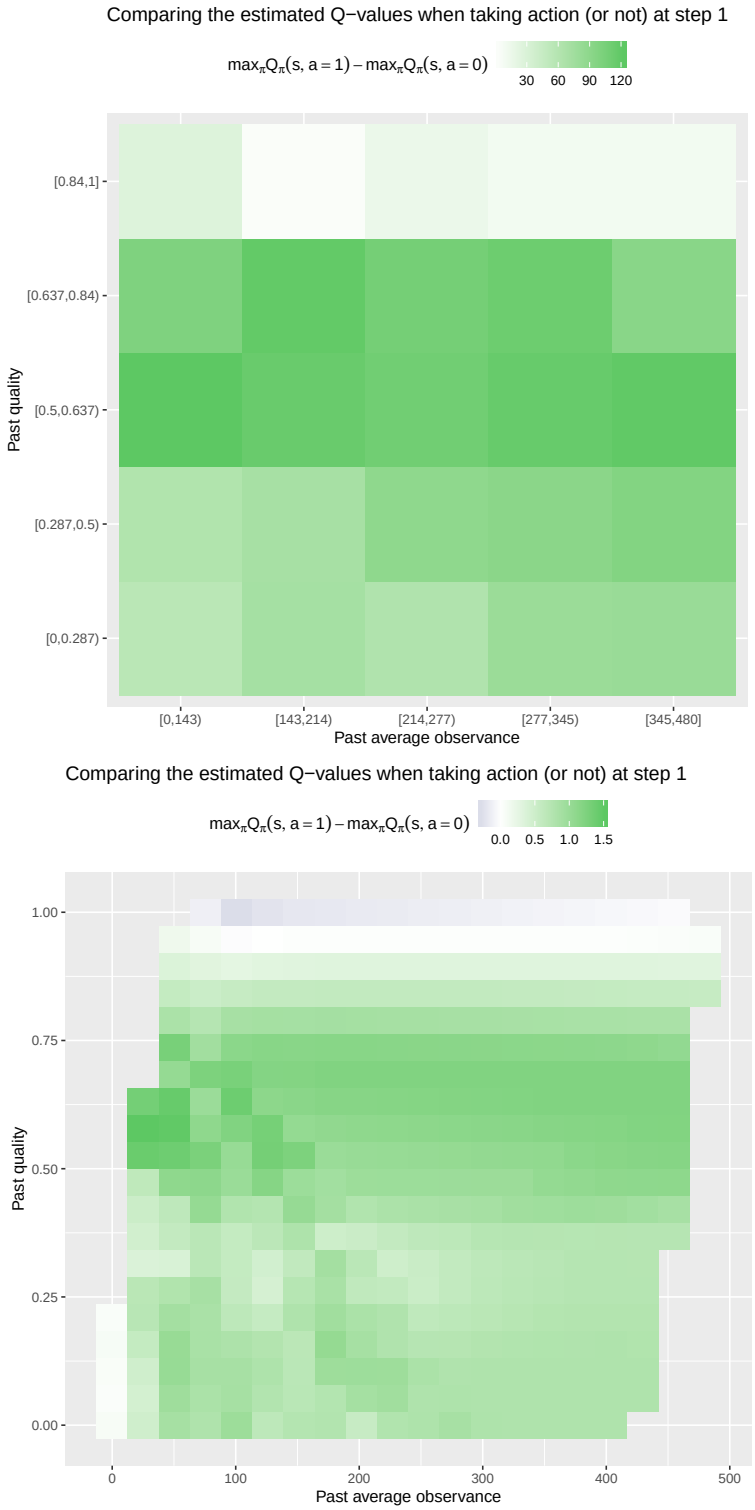

Figure 8.2: Differences in estimated Q-values when taking an action at step 1 or not taking the action, positive indicating it is favorable to take the action. Upper graph: the estimation is based upon dynamic programming relying on a discretization of the state space. Lower graph: the estimation is on neural network approximation.

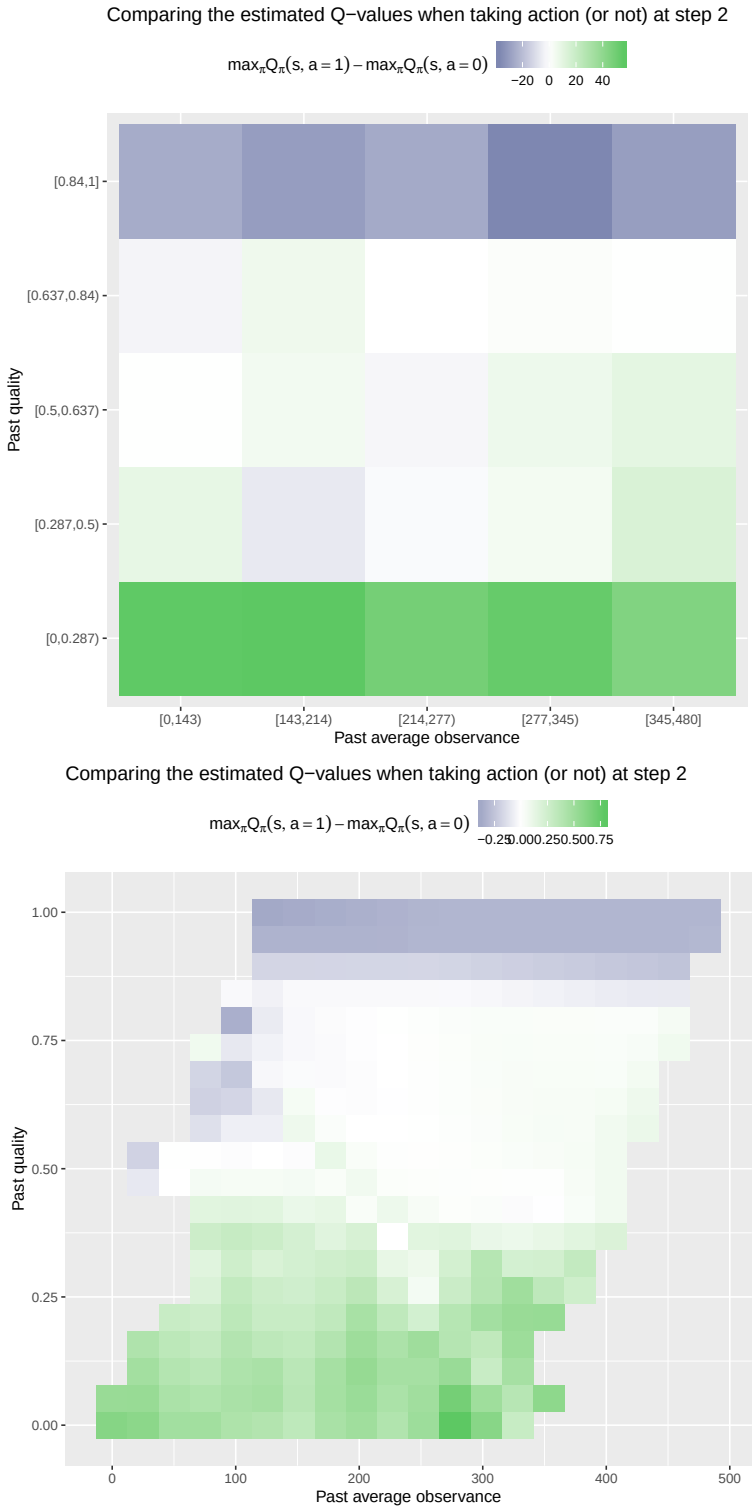

Figure 8.3: Differences in estimated Q-values when taking an action at step 2 or not taking the action, positive indicating it is favorable to take the action. Upper graph: the estimation is based upon dynamic programming relying on a discretization of the state space. Lower graph: the estimation is on neural network approximation.

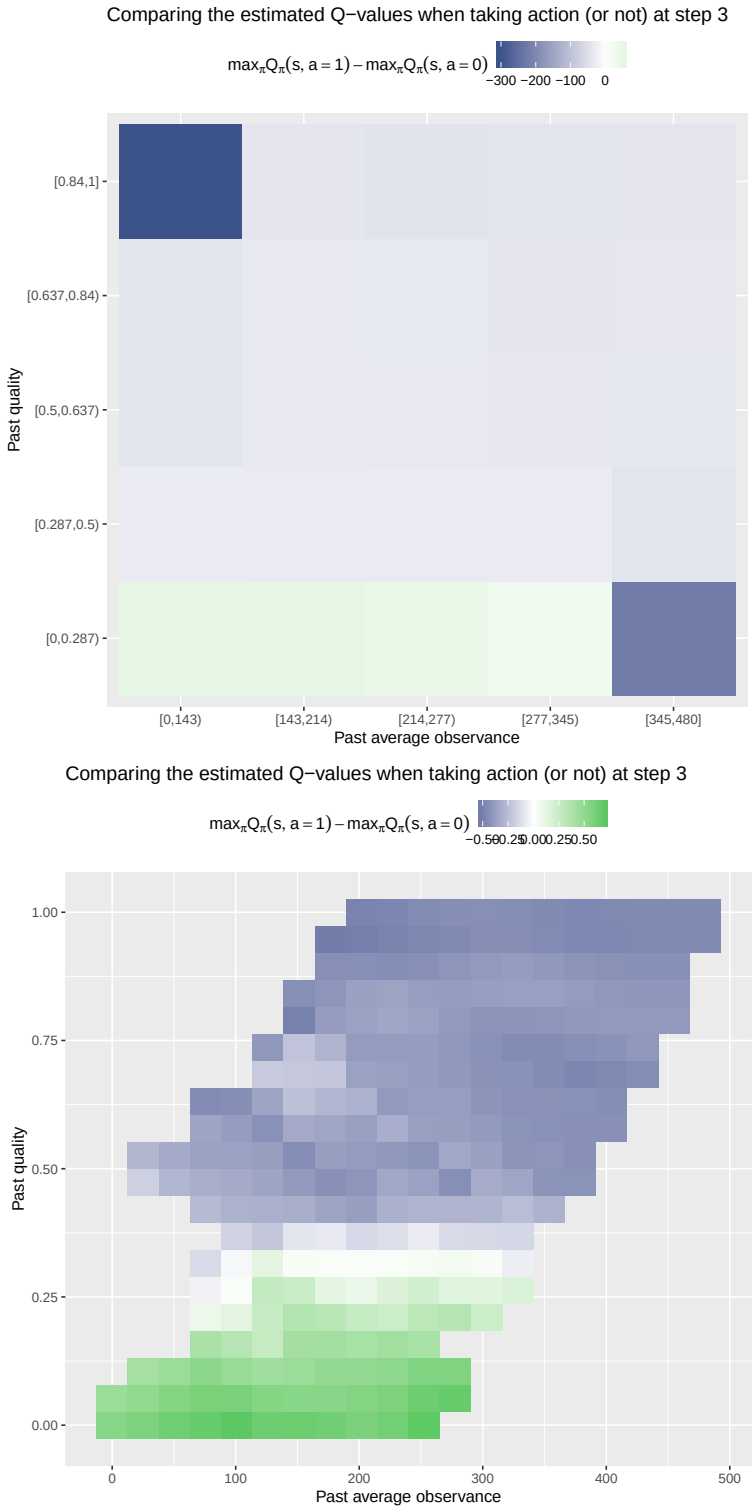

Figure 8.4: Differences in estimated Q-values when taking an action at step 3 or not taking the action, positive indicating it is favorable to take the action. Upper graph: the estimation is based upon dynamic programming relying on a discretization of the state space. Lower graph: the estimation is on neural network approximation.

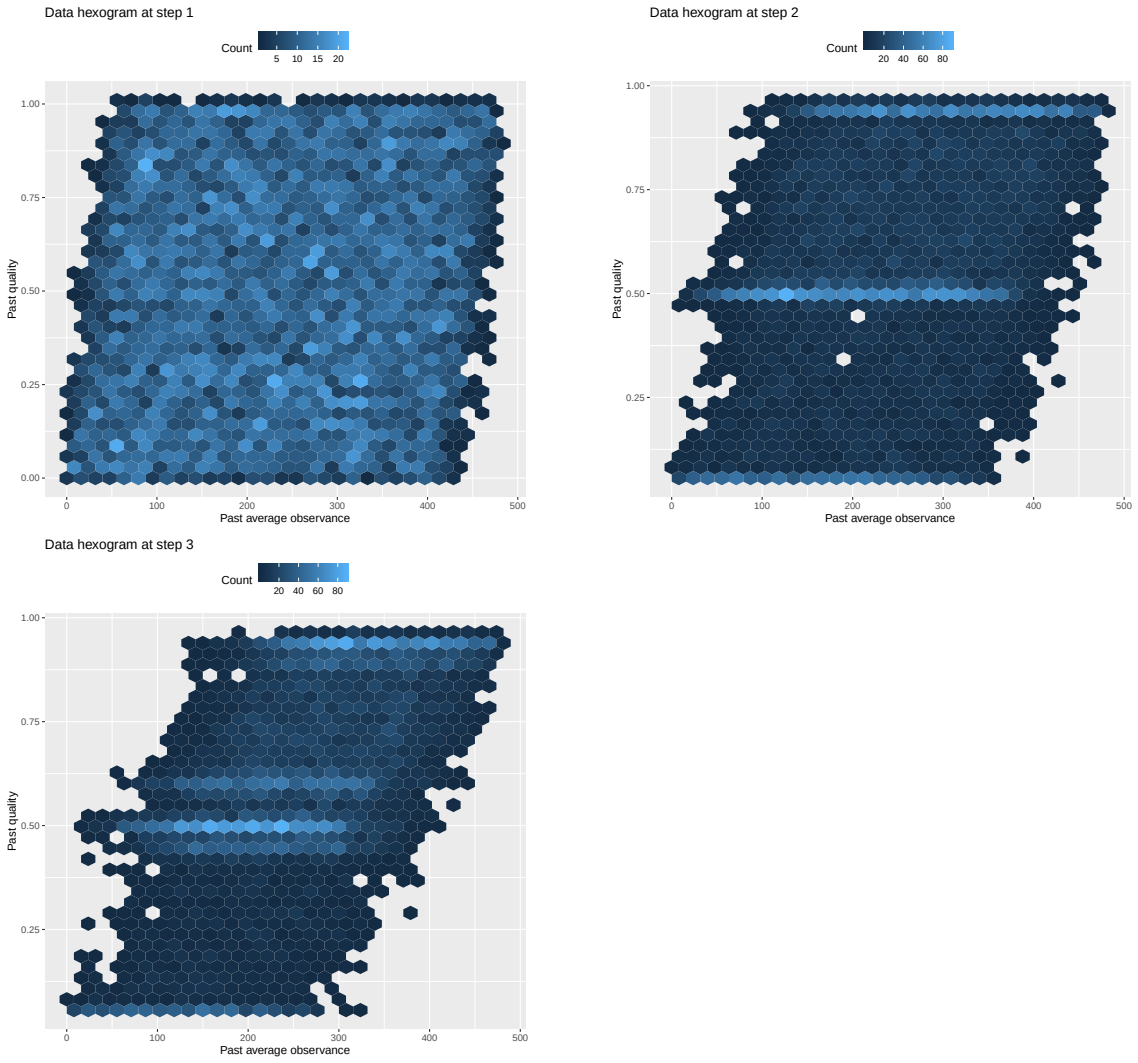

Figure 8.5: Data distribution under state space

8.4 Conclusion

In this chapter we studied an episodic sequential decision-making problem, where at each milestone, we could either take an action or not take it in hope of making our patient observant with the equipment we have provided them with. The overall objective is to maximize the patient's observance and, if our efforts seem to not be working for a particular patient, or if the patient is already quite observant, probably focus our attention on patients more in need of assistance. In the instantiation considered, we showed how classic value-based approaches, either focusing on a discretized state space or using a continuous function approximation, can take well-performing decisions based on past observed data. This work was used for an internal Proof of Concept at Air Liquide.

Chapter 9

Intelligent Questionnaire †

Summary

Data collection can be detrimental to user/customer/patient experience if not handled properly. While many issues are posed during data collection, such as question framing and ordering, we consider the construction of a questionnaire constrained in its length and ideally adaptive to the user/customer/patient: out of a set of p informations we would hope to retrieve, the algorithm will select sequentially q questions, adapting its choices based on initial answers and potentially initial informations. Once retrieved, the partial information $\tilde{X}$ will be used as predictor for unknown target Y and so the quality of this adaptive questionnaire built will be judge on how accurately target Y can be predicted from the partial information. Several works from the literature point to knowledge-based approaches and a few to statistical approaches. We frame this sequential decision-making problem as a Markov Decision Process where the state variable represents the partially-known vector, the actions are the questions we ask and the reward represents the prediction performance whenever we choose to stop asking our questions. Exploiting the hierarchical structure of the problem, we apply (approximate) dynamic programming backward rule to find the optimal result from fixed data batches. The effectiveness of the approach is demonstrated on three toy models and three benchmark datasets (in supervised learning) against two baselines: a constrained CART tree and a best q subset approach. The extension to the online setting is experimented in detail for one of the toy models.

9.1 Context

In user interaction, less is often more. When asking questions, it is desirable to ask as few questions as possible or given a budget of questions asking the most interesting ones. We study the case of smart questionnaire in which the questions asked may depend on the previous answers. More precisely, we consider a set of p questions (asking for elements of X) in a prediction context (target Y based on X). Given a budget of q questions, we design an algorithm choosing sequentially the next question to be asked so that the predictive power is maximized after having q answers assuming we have observed the whole set of answers on a first dataset. No prior knowledge is available. An illustrative example of the expected result is given in figure 9.1.

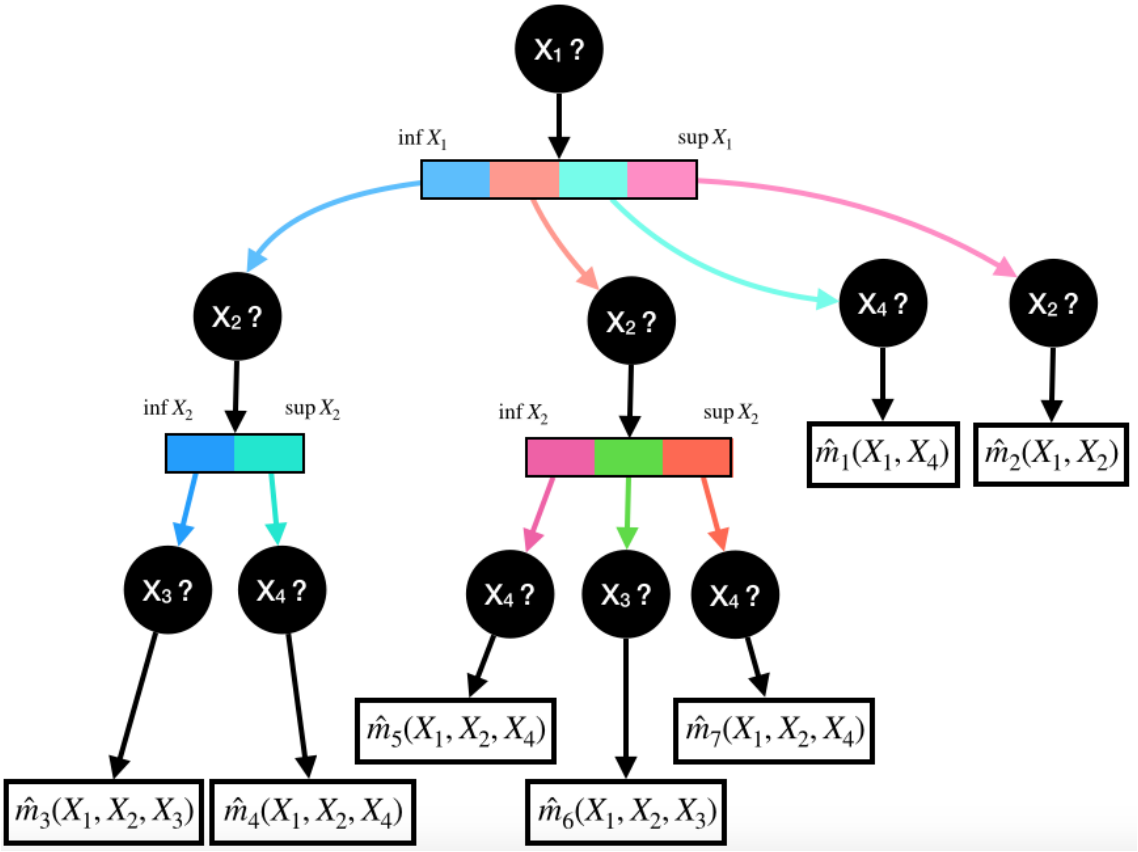

Figure 9.1: Illustrative example of smart questionnaire obtained: without any apriori information, we start by asking for the value of X_1. Then, depending on the value obtained, we will either ask for the value of X_2 or X_4. If X_1 was high enough (cyan and pink cases) we stop the algorithm and we have prediction function $\hat{Y}_1$, resp. $\hat{Y}_2$, based on the actual values of couple (X_1, X_4), resp. (X_1, X_2). On the left side of the tree, blue and red cases, we ask for X_2 and depending on the values of couple (X_1, X_2) we will ask for the value of X_3 or X_4 and then give a prediction. One can view it as a continuous version of the decision tree, where we ask for the value of a single feature sequentially rather than applying univariate thresholds, which are much less informative. Also, the final prediction is based on the features retrieved, and so exploits the complete dataset, rather than only similar observations as decision trees do. In this example we went to a depth 3 maximum, and stopped the algorithm prior to maximum depth for nodes on the right.

9.1.1 Motivation

We want to choose sequentially out of the p elements of X in order to give an accurate prediction of Y whilst not collecting all the elements. This approach is relevant in cases where the cost of getting the data is not negligeable in front of inaccurate predictions: either gathering the data requires a lot of money, a lot of time or is tiring for the responder. Note that this is quite a different setting than in classic supervised learning, where the data's price is (almost completely) disregarded.

This setting is quite general and comprises for example:

- Patient follow-up. Consider a patient who is hospitalized at home and fills in a daily checkup questionnaire asking for leg pain, a hurting chest or physical discomfort. The aim of the questionnaire being to check the status of the patient, we would like to ask the most pertinent questions as soon as possible and personalize the questions to their status.
- Prospective calls e.g. telemarketing. Instead of bluntly unrolling the same list of questions, we could adapt our series of questions in order to know as fast as possible if the person called would be interested or not.
- Cold start issue with new customer. When subscribing to a new service, it is not uncommon to get asked some questions in order to personalize the service (e.g. Netflix, web service provider). Assuming we already have a customer clustering at hand, we would like to find the newcomer's cluster with as little questions possible (so as to not "bore the patient out").
- Balancing acquisition cost/available information. Assuming the data is paid for (personal data sold) we would choose which information to pay for for each people.
- Storing less data to make as good predictions. Considering that data storage has non-negligeable cost (financially, facility-wise, environmentally), we wish to only keep data essential to the prediction, storing a sparse matrix would suffice.

9.1.2 Detailed context

Consider the pair of random variables (X, Y) where :

- $Y \in \mathcal{Y}$, $\dim(\mathcal{Y}) \geq 1$; Y is a variable of interest
- $X \in \mathcal{X} = \otimes_{j=1}^{p} \mathcal{X}_i$, $p > 1$, $\dim(\mathcal{X}_i) = 1$; X is a variable which can be used to predict Y.

In supervised learning, the task is to build a predictor of Y given X e.g. its conditional distribution, moment(s) or quantile(s). Here, we are interested in the way that vector X is collected: in a classic questionnaire, we would go through each element of X in a pre-specified order or at random in order to avoid bias issues. Since we collect X in order to predict Y we would like to build an intelligent questionnaire which would collect elements of X which are the most useful for the prediction task. As such, this questionnaire will take into account the realizations of the elements of X that were already requested and check which new feature could be the most useful for our task. The questionnaire process is described in figure 12 and here below we introduce important notations.

Consider $\tilde{\mathcal{X}}^{\text{full}} \stackrel{\Delta}{=} \otimes_{j=1}^{p} (\mathcal{X}_i \cup \{\text{“unknown”}\})$ the space of partially known feature vector X and $\tilde{\mathcal{X}}$ its version restricted to the vectors whose antecedents did not stop the questionnaire process,

which in our case corresponds to the vectors where at most q elements are known:

$$\tilde{\mathcal{X}} \triangleq \{\tilde{x} \in \tilde{\mathcal{X}}^{\text{full}} : \mathbf{card}\{j \in \{1,\dots,p\} : \tilde{x}_j \neq \text{“unknown”}\} \leq q\}. \tag{9.1.1}$$

Accordingly we define *stop* function as follows:

$$\forall \tilde{x} \in \tilde{\mathcal{X}}\ \mathit{stop}(\tilde{x}) = \begin{cases} \texttt{true} & \text{if } \mathbf{card}\{j \in \{1,\dots,p\} : \tilde{x}_j \neq \text{“unknown”}\} \geq q \\ \texttt{false} & \text{otherwise} \end{cases}.$$

We will design algorithm π which to any masked feature vector $\tilde{x} \in \tilde{\mathcal{X}}$ assigns a value in the set $\mathcal{A} \triangleq \{1,\dots,p\}$, indicating which feature element should be recovered. This function will allow us to ask questions in an adaptive, and hopefully, intelligent manner, quantified by the *score* function, until our question budget is fully used. On the notations: the set of questions asked will be denoted by A_t, t being the step of the questionnaire, starting at 0. The available knowledge at time t will be written $\tilde{X}_t$ and its j-th element will be written $\tilde{X}_{t,j}$.

We assume, without loss of generality, that the more available information on X the better. Also, we assume when a question is asked, an element of X is revealed to us and we trust it to be accurate. As such, the same question is never asked twice and we completely trust the response given.

Algorithm 12: Generic intelligent questionnaire algorithm process. Depending on whether we are collecting data with people volonteering to inform us of the whole information or not, we consider both returns.

Initialize $\tilde{X}_0 =$ “unknown”^{p}
Interacting individual knows $(X, Y) \in \mathcal{X} \times \mathcal{Y}$
Input algorithm $\pi : \tilde{\mathcal{X}} \to \{1,\dots,p\}$
Input stopping criterion $\mathit{stop} : \tilde{\mathcal{X}} \to \{\texttt{false}, \texttt{true}\}$
Input scoring function $\mathit{score} : \tilde{\mathcal{X}} \times \mathcal{Y} \to \mathbb{R}_+$
for $j \in \{1,\dots,q\}$ **do**
    `/* Select most pertinent question to ask */`
    $A_t = \pi(\tilde{X}_t)$
    `/* Observe response */`
    $\tilde{X}_{t+1} = \tilde{X}_t\ ; \quad \tilde{X}_{t+1,A_t} = X_{A_t}$
    `/* Decide whether to stop */`
    if $\mathit{stop}(\tilde{X}_{t+1})$ **then**
        break
    end
end
if full_information_mode **then**
    return $\{X, Y, (A_l)_{l=0}^t, \mathit{score}(\tilde{X}_{t+1}, Y)\}$
else
    return $\{(A_l)_{l=0}^t, \hat{m}(\tilde{X}_{t+1})\}$
end

9.1.3 Mathematical objective

Let Π denote the set of functions mapping $\tilde{\mathcal{X}}$ to $\mathcal{A}$ i.e. the set of acceptable questionnaire algorithms. Note that, assuming more available information is better, we should only consider algorithms which do not ask twice or more the same questions. In all generality, we focus on solving the following optimization problem:

$$\forall \tilde{x} \in \tilde{\mathcal{X}} \qquad \pi^*(\tilde{x}) = \arg\max_{\pi \in \Pi} \mathbb{E}_{\pi,(X,Y)} \left[score(\tilde{X}_q, Y) | \tilde{X} = \tilde{x} \right] \tag{9.1.2}$$

where $\tilde{X}_q$ is the partially-known feature vector obtained when the algorithm is stopped (q questions asked). The score function will typically take into consideration our ability to accurately predict target Y based on the partial information retrieved. We propose the following score function:

$$\forall \tilde{x} \in \tilde{\mathcal{X}}, y \in \mathcal{Y} \quad score(\tilde{x}, y) = -\mathbf{L}(\hat{m}(\tilde{x}), y) \tag{9.1.3}$$

where $\hat{m}$ is a prediction function of the target based on partial information and $\mathbf{L}$ is an individual risk measure, such as the squared error in regression or the log-loss in classification.

As π^* depends on the final feature vector reached, one can immediately see that depending on the stopping function implemented *stop*, the optimal solution of the problem above changes.

9.1.4 Available data

We assume that we have at our disposal a set of n, $n \gg p$, independent realizations $(x^{(i)}, y^{(i)})_{i=1}^n$ of couple (X, Y) following the, unknown to us, joint distribution $\mathbb{P}(X, Y)$. This dataset will be used to learn prediction function $\hat{m}$ and policy π^*.

9.2 Bibliography

In this section we present baselines to learn π^* (or less optimal version, if restrictions on Π are applied), literature on adaptive questionnaires, decision trees, planning and reinforcement learning.

9.2.1 Baselines

All possible solution

Assuming we are in case 1 or case 2 (referring to the Three degrees of difficulty comment), we could imagine that we could solve the problem by going through all possible trees (so an adaptive solution). For case 1 it is quite straightforward, for case 2, we would just need to adapt estimates based on the answer to the first question with continuous response. This brute force aproach, when available i.e. computationnally tractable, serves as an important baseline.

Best variable subset selection

Another approach to solve the problem is to find the best subset of variables, without considering an adaptive sequence of questions.

Let us write unmasked the function which to an element x of $\mathcal{X}$ and a subset s of $\{1, \ldots, p\}$ maps its unmasked version $\tilde{x} = (\tilde{x}_j)_{j=1}^p$ such that $\tilde{x}_j = x_j$ if $j \in s$ and $\tilde{x}_j =$ "unknown" otherwise.

For any choice of $q \in \{1, \dots, p-1\}$, let $subsets(q)$ denote the complete set of subsets of q elements from set $\{1, \dots, p\}$. If the dimension of $subsets(q)$ allows it, one can attempt to solve:

$$subset^* \triangleq \arg \max_{s \in subsets(q)} \mathbb{E}_{(X,Y)} \left[score(\text{unmasked}(X, s), Y) \right]. \tag{9.2.1}$$

An alternative approach is to compute individual prediction scores for each of the p features and rank the questions based on that score, such as one would get with variable importance for examples. This approach, as it does not take into account the score function is clearly less performant, and it is non adaptative to the user's already filled answers, hence our approach will be more generic.

In any case, fixed-subset is a straightforward and therefore important baseline.

9.2.2 Related works

Adaptive questionnaires have been investigated through knowledge-based approaches in several fields amongst which we find e-learning Nokelainen et al. [2001] and healthcare Dunlop [2019]. In Mwamikazi et al. [2014] the authors investigated an approach relying on association rules for the prediction and question selection tasks and experimented their algorithm on Myers-Briggs tests. Association rules only cover binary features unfortunately.

Decision tree algorithms such as CART Breiman et al. [1984] are by construction adaptive questionnaires built for prediction. As we will further discuss, a tree of depth q provides a solution but is optimized in a top-down manner (greedy optimization) and the "decisions" consists of single coordinate thresholding which eases the computational aspect, but is evidently suboptimal.

This problem is a sequential decision-making problem and can therefore be represented by a Markov Decision Process Puterman [2014] where the state is the information currently available, actions are the questions we ask and as final rewards the prediction performance based on the final partial information. Such a modeling has been used for instance in Chen et al. [2018] to tackle the game of 20 questions relying on a smart matrix factorization and more recently in a health diagnosis problem using a reinforcement learning approach Besson et al. [2018].

9.2.3 Association rules

A baseline for studying basket item relationships is association rules, the idea being to observe how sets tend to appear with each other through different measures such as support, the proportion of transactions containing both item A and B, the (directed) confidence, the proportion of cases where B appears given that A does and the lift, ratioing the confidence with the proportion of transaction containing item B. All those proportions and ratios are heuristics to help algorithms either recommend new items or understand item relationships. As such, it can be adapted to the supervised learning setting quite easily and to our problem in particular, as was proposed for example in Mwamikazi et al. [2014]. Association rules only are used to cover binary features, although, with a probabilistic model approach it could probably be extended.

9.2.4 Decision trees

Decision trees, as presented in part I, are built to ask binary questions in order to predict a target variable e.g. $Q_1 = \mathbb{1}\{X_1 < \tau\}$. There are several implementations of decision trees in

R and Python languages, relatively straightforward to understand how they work and even to recode them. Although it does not solve the problem we are interested in, they represent a proper baseline for us to compare to. Also, compared to the variable importance approach presented above, decision trees are at least adaptive to the answers revealed to them.

Below are the shortcomings.

First, from their construction, the decision tree asks whether a feature is below a threshold or not, which is different from asking for the value of a feature, except in the binary case. This implies that for every non-binary feature, we should consider that one the decision tree has asked a question on it, it is known, and the following splitting can be done without cost of question. This is a small code tweak, but it recalls that the construction of the decision tree does not exploit to the full the knowledge from non-binary features.

Second, and related to the first issue, the decision trees are built by optimizing the split ahead. The global hierarchy is not fully optimized for the final prediction accuracy, unless of course we consider a tree of depth one.

Third, and last shortcoming (probably the least essential though), existing implementations do not allow users to define their accuracy function themselves, which might be an inconvenience in the case where Y or X aren't categorical and we can't simply assign some class weight.

One of the shortcomings of decision trees as stated previously is that classic algorithms such as CART and C4.5 calibrate splits by solving local optimization problems and so the final tree (independently of the pruning process) is likely suboptimal. Several efforts have been made in order to optimize the decision tree at a global scale.

In Bennett [1995], the authors use Linear Programming with a fixed decision tree shape to learn the tree parameters which optimize globally the decision tree, rather than local optimizations. It seems that this approach is limited by the program complexity as well as the tree structure imposed.

Many other approaches to the greedy top-down approach have been tested, among which: Integer Programming, as in Günlük et al. [2018], Constraint Programming, as in Verhaeghe et al. [2019] and Evolutionary Algorithms, as in Barros et al. [2013].

More recently, in Nunes et al. [2020] the authors rely on Monte-Carlo Tree Search (MCTS), a quite advanced technique to optimize the decision tree globally rather than locally. MCTS uses Upper Confidence Tree algorithm combined with Monte-Carlo (rolling out the tree for several iterations). Even more recently, in Aglin et al. [2020], the authors take an approach from Operations Research community (DL8 - using branch and bound) and combine it with an upper bound estimate, similarly to the UCB approach.

9.2.5 Planning & Reinforcement Learning

The problem we present is a sequential decision-making problem, hierarchical as features are throughout the steps unveiled and episodic as the number of steps is set to q. An appropriate mathematical model is a Markov Decision Process Puterman [2014], where the state variable describes the available information, the actions are which information is asked for and the reward is the final expected *score* as in the definition of π^* (eq 9.1.2). The only remaining question is how to solve this optimization problem from the large sample of techniques from the planning/Reinforcement Learning literature, see for instance Sutton and Barto [2018] or Li [2018]. Note that Reinforcement Learning methods are rooted on the same theory than the planning" methods, but extend to the case where data does not come in a single batch, but are acquired sample after sample, and the

cost of the data is considered not negligeable either from cost/time perspective or simply because the space dimension is too large for full exploration. The Deep Reinforcement Learning methods, combining Reinforcement Learning theory and the generalization power of deep neural networks for function approximation allows to handle easily large and/or continuous state and action spaces. As such, it is common in Reinforcement Learning to design an algorithm which balances (a) the data collection (i.e. which states are visited, exploration) and (b) the actual task of interest - exploitation. As related works linked to the adaptive questionnaire issue we may cite the works of Chen et al. [2018], which tackle the game of 20 questions relying on a smart matrix factorization and Besson et al. [2018] which study a health diagnosis problem using a reinforcement learning approach.

9.3 Methodology

In this section we introduce Markov Decision Processes in more detail, we present Approximate Dynamic Programming, the approach we propose to solve 9.1.2, we talk about exploration approaches for the Reinforcement Learning problem extension and discuss choices for function *score*.

9.3.1 Markov Decision Process

The questionnaire process can be modeled by a Markov Decision Process (MDP, Puterman [2014]) $\mathcal{M} = (\tilde{\mathcal{X}}, \mathcal{A}, \mathbf{T}, \mathbf{R})$ where $\tilde{\mathcal{X}}$ is the state space, where partially-known feature vector $\tilde{X}$ lives and $\mathcal{A} = \{1, \dots, p\}$ is the action space, the indexes of feature elements we can collect. $\mathbf{T}$ is the transition function mapping $\tilde{\mathcal{X}} \times \mathcal{A} \times \tilde{\mathcal{X}}$ to $[0, 1]$ defined as

$$\forall \tilde{x} \in \tilde{\mathcal{X}}, j \in \mathcal{A}, \tilde{x}' \in \tilde{\mathcal{X}} \qquad \mathbf{T}(\tilde{x}, j, \tilde{x}') \triangleq \mathbb{P}_X(X_j = \tilde{x}'_j \mid \bigcap_{\{l:\tilde{x}_l \neq \text{“unknown”}\}} \{X_l = \tilde{x}_l\}) \tag{9.3.1}$$

if $\tilde{x}_j =$ “unknown”, $\tilde{x}'_j \neq$ “unknown” and $\forall l \neq j, \tilde{x}_l = \tilde{x}'_l$. If however $\tilde{x}_j \neq$ “unknown” and $\forall l \in \{1, \dots, p\}, \tilde{x}_l = \tilde{x}'_l$, the question has already been asked so this probability is 1 and otherwise this probability is 0 as it indicates some incompatibility between $\tilde{x}$ and $\tilde{x}'$. Finally, $\mathbf{R}$ is the reward function, such that for any state $\tilde{x}$ such that $stop(\tilde{x}) =$ `true`

$$\mathbf{R}(\tilde{x}, ., .) \sim \mathbb{P}_{(X,Y)}[score(\tilde{x}, Y) | \tilde{X} = \tilde{x}] \tag{9.3.2}$$

and equals 0 otherwise.

Start state The MDP starts with initial state $\tilde{X}_0 =$ “unknown”^{p}, we then ask question A_0, coordinate A_0 is revealed (state $\tilde{X}_1$), we then ask question A_1, reach state $\tilde{X}_2$, and so on and so forth, until a terminal state is reached (*stop* function).

Policy π is a function which, given a state $\tilde{x} \in \tilde{\mathcal{X}}$ assigns an action $a \in \mathcal{A}$. Typically, we will enforce to choose an action a such that $\tilde{x}_a =$ “unknown” in the idea that the more information the better, and that priorly required information is to be trusted.

From (X, Y) data to RL batches From the initial dataset $(x^{(i)}, y^{(i)})_{i=1}^n$. From there we can directly create the set of transitions and the set of rewards for terminal states, since we define

function *score*: (1) the set of rewards for terminal states

$$\Big(\text{unmasked}(x^{(i)}, s), score(\text{unmasked}(x^{(i)}, s), y^{(i)})\Big)_{\substack{i=1,\dots,n \\ s \in subsets(q)}}$$

which will be used to learn the reward function and (2) the set of transitions from state to state

$$\Big(\text{unmasked}(x^{(i)}, s), j, \text{unmasked}(x^{(i)}, \{s \cup j\})\Big)_{\substack{i=1,\dots,n \\ s \in subsets(0,\dots,q-1)}}.$$

to learn transition function. The score function will be defined by us so it is easy to get this data.

Approximate dynamic programming Because the problem is episodic and hierarchical, one approach to solve equation 9.1.2 is to sequentially compute functions function approximations of optimal state value functions at layers, going backwards, as presented in section REF. If the feature space $\mathcal{X}$ is discrete and let if $n \gg support(\mathcal{X})$, estimated functions can simply be conditional means. Now, if the feature space is continuous, or mixed discrete-continuous, the most straightforward approach is to use neural networks as approximators (Approximate Dynamic Programming Bertsekas et al. [1996]).

Non-greedy decision-making To handle exploration, we consider the softmax and upper-confidence bound approaches presented in section ... For the ucb exploration bonus, we used the count of previous couple or triplets already observed before.

9.3.2 Scoring

As we stated before, we consider the following score function:

$$\forall \tilde{x} \in \tilde{\mathcal{X}}, y \in \mathcal{Y}, \quad score(\tilde{x}, y) = -\mathbf{L}(\hat{m}(\tilde{x}), y)$$

where the prediction function $\hat{m}$ takes as input a partially-known feature vector $\tilde{x} \in \tilde{\mathcal{X}}$ and outputs a prediction of Y within $\mathcal{Y}$ and the loss function $\mathbf{L}$ takes as input the couple $(\hat{y}, y)$ and outputs a quantity indicating how bad is the prediction $\hat{y}$ with respect to the value y.

Evidently, the prediction function $\hat{m}$ should be learnt with a loss function similar to $\mathbf{L}$, so that their action is not antagonist, see part I for supervised learning algorithms and appropriate loss functions.

Quantifying predictive power Please note that, for a relatively cheap price, we can build an estimate of

$$\mathbb{E}_{(X,Y)}[\mathbf{L}(\widehat{m}(\tilde{x}), Y) | \tilde{X} = \tilde{x}] \tag{9.3.3}$$

for any vector $\tilde{x} \in \mathcal{X}$. This quantity could be very useful in order to (a) provide information on the final prediction provided (prediction quality) and (b) if we consider that what matters is getting precise estimates, we could stop the algorithm when sufficiently accurate estimates are obtained, relaxing the budget constraint.

9.3.3 Construction of predictor $\widehat{m}$

We need to construct a prediction algorithm $\widehat{m}$ which takes as input an element of $\tilde{\mathcal{X}}$ and outputs some prediction of interest of $Y|\tilde{X}$ e.g. mean, distribution.

We consider four different approaches:

1. build a supervised learning algorithm for every possible q subset of answered questions
2. build a single supervised learning algorithm where unknown components have specific values and as input we also have whether the component is missing or not : neural networks seem to be appropriate, restricted random forests could also work (need to keep the mask information at all times)
3. imputation of the yet unknown data with supervised learning algorithm on all features: based on the available dataset, we could apply imputation scheme to retrieve potential full set of information
4. instance-based approach: akin to the imputation approach, defining one dimensional kernels to compare known components to the rest of the database

In general, we will focus on the first approach, building for each subset a specific supervised learning algorithm. However in some cases, typically when the features are binary and n is sufficiently large, the instance-based approach can be quite interesting relying on a kernel approach, so let us give some more details.

Assuming $\hat{m}_0$ is a supervised learning algorithm mapping $\mathcal{X} \to \mathcal{Y}$, we could define $\hat{m}$ as

$$\forall \tilde{x} \in \tilde{\mathcal{X}} \quad \widehat{m}(\tilde{x}) = \frac{\sum_{i=1}^{n} \widehat{m}_0(x^{(i)}) \mathbf{N}(x^{(i)}, \tilde{x})}{\sum_{i=1}^{n} \mathbf{N}(x^{(i)}, \tilde{x})} \tag{9.3.4}$$

where $\mathbf{N} : \mathcal{X} \times \tilde{\mathcal{X}} \to \mathbb{R}_+$ is a closeness measure e.g. radial function:

$$\forall (x, \tilde{x}) \in \mathcal{X} \times \tilde{\mathcal{X}} \qquad \mathbf{N}(x, \tilde{x}) = \exp\{-\sum_{j=1}^{p} \frac{(x_j - \tilde{x}_j)^2}{\sigma_j^2}\} \tag{9.3.5}$$

where $(\sigma_1, \ldots, \sigma_p)$ is a set of non-negative reals controlling the scaling of each dimension independently.

9.3.4 Process summary

If we recapitulate the process of building the smart questionnaire, it leads to the following steps:

1. Collect data samples and prepare the state-action-next state triplets as well as final rewards set
2. Choose supervised learning family $\mathcal{F}$ and calibrate $\hat{m}$ based on the available data
3. Choose risk metric $\mathbf{L}$ (probably a good idea to think about it before calibration of $\hat{m}$)
4. Choose stopping criterion *stop* depending on what is most important: limiting the number of questions overall, stopping when we believe we have enough accuracy, etc
5. Learn π^* based on *score*, $\hat{m}$, *stop* with approximate dynamic programming.

9.4 Experiments

To evaluate the performance of our approach, we built three toy models and considered three benchmark datasets for the regression task. For each of those six problems, we built a smart questionnaire algorithm with a budget of $q = 3$ features to uncover based on training data. The training data was split in three equal parts: one to train the final predictor $\hat{m}$, one to train the smart questionnaire, and finally one to validate the smart questionnaire training. We used `R` package `keras` to calibrate the feed-forward neural networks, relying on `rmsprop` optimizer, learning rate reduction on plateau as well as early stopping. Problem specific details, such as network dimensions can be found on the following Github repository github.com/FredericLoge/SmartQuestions.

In the first two subsections we present in details the six cases considered. Subsequently we present the results obtained in the planning setting for each of those problems: the performances of our approach (see table 9.4) compared to best subset and CART approaches, as well as to an oracle, which has access to all p features. We also present some representations of the intelligent questionnaire computed. Finally, we explore on one of the toy models the online version of the algorithm, considering different exploration policies for data collection, evaluating the variability of the approaches in terms of final as well as cumulative performances (one could mention *regret*).

9.4.1 Toy models

Three toy models were considered in order to test our approach. In each case we simulated in total 6000 samples, 67% of which are used for training and the rest for testing. As predictor functions, we used random forests with 100 trees, as implemented in the `R` package `randomForest`. For models #2 and #3, we will write ε a standard gaussian noise generated independently of features X. For the first toy model we considered the inaccuracy score function and for the two models following we used the absolute prediction error.

Model #1, set of rules with binary covariates

We consider $p = 8$ mutually independent binary features

$$X_j \sim \text{Bernoulli}(0.5) \quad \forall j \in \{1, \dots, p\}.$$

Let $E(X)$ denote the union of arbitrarily chosen events:

$$\begin{aligned} E(X) =& \{X_1 = X_2 = X_8 = 0\} \cup \{X_6 = 0, X_2 = X_3 = 1\} \\ & \cup \{X_8 = X_1 = X_3 = 1\} \cup \{X_4 = X_5 = X_6 = 0\} \\ & \cup \{X_3 = X_4 = X_2 = 1\} \cup \{X_4 = X_8 = X_1 = 1\} \\ & \cup \{X_3 = X_5 = X_7 = 0\}. \end{aligned}$$

Then, $Y|X = \mathbb{1}\{\bar{E}(X)\}$.

Model #2, set of rules with binary and one continuous covariate

We consider $p = 6$ mutually independent features

$$\forall j \in \{1, \dots, p-1\} \quad X_j \sim \text{Bernoulli}(0.5), \quad X_p \sim \mathcal{U}[0, 1].$$

Consider $E_1(X)$ and $E_2(X)$ the arbitrary set of events defined as follows:

$$E_1(X) = \{X_1 = 0, \{X_2 = 0 \cup X_6 > .7\}\} \cup \{X_4 = X_5 = 0, X_6 > .4\} \cup \{X_1 = X_3 = 0, X_6 > .8\},$$
$$E_2(X) = \{X_1 = 1, \{X_3 = 1 \cup X_6 > .7\}\} \cup \{X_3 = X_5 = 1, X_6 > .6\}.$$

Now,

$$Y = m(X) + .2\varepsilon \quad \text{where } m(X) = \begin{cases} 0 & \text{if } \bar{E_1}(X) \cap \bar{E_2}(X) \\ 1 & \text{if } E_1(X) \cap \bar{E_2}(X) \\ 2 & \text{if } E_2(X) \end{cases}.$$

Model #3: regression with continuous covariates

We consider $p = 8$ mutually independent random gaussians features

$$X_j \sim \text{Gaussian}(0, 1) \quad \forall j \in \{1, \dots, p\}.$$

The target variable is defined by the following model $Y = m(X) + \sigma\varepsilon$ where

$$m(X) = \begin{cases} X_1 + X_2 & \text{if } X_3 < 0 \\ X_4 + X_5 & \text{otherwise} \end{cases}$$

and $\varepsilon \sim \text{Gaussian}(0, 1)$ is the noise random variable, independent of X. Parameter σ controls the noise level in the model. The signal level is constant:

$$\begin{aligned} \mathbb{V}((X_1 + X_2)\mathbb{1}\{X_3 < 0\} + (X_4 + X_5)\mathbb{1}\{X_3 \geq 0\}) &= \mathbb{V}((X_1 + X_2)\mathbb{1}\{X_3 < 0\}) + \mathbb{V}((X_4 + X_5)\mathbb{1}\{X_3 \geq 0\}) \\ &= \mathbb{V}(X_1 + X_2)\mathbb{P}\{X_3 < 0\} + \mathbb{V}(X_4 + X_5)\mathbb{P}\{X_3 \geq 0\} \\ &= 2 \cdot 1/2 + 2 \cdot 1/2 \\ &= 2 \end{aligned}$$

hence for a given sound to noise ratio ($\mathbf{snr} = \mathbb{V}(m(X))/\mathbb{V}(\sigma\varepsilon)$), we can pick $\sigma^2 = 2/\mathbf{snr}$. Assuming we are interested in estimating perfectly $\mathbb{E}[Y|X] = m(X)$, and we measure it by R^2 metric, we would hope to get:

$$R^2_{high} = \frac{\mathbb{V}(m(X))}{\mathbb{V}(m(X) + \sigma\varepsilon)} = \frac{\mathbf{snr}}{1 + \mathbf{snr}}$$

since $\mathbb{V}(m(X)) = \sigma^2\mathbf{snr}$ by definition of $\mathbf{snr}$. In our planning experiments, we wil take $\mathbf{snr} = 1$ and so $\sigma = \sqrt{2}$, which is a lower bound on prediction performance i.e. one would need to ask the three right elements and have a perfect model.

9.4.2 Benchmark datasets

Three benchmark datasets where considered: the Boston Housing dataset Harrison Jr and Rubinfeld [1978], the more recent AMES dataset De Cock [2011] and lastly a Coronary Heart Disease (obtained from Kaggle: https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset, initial source claimed: Framingham Heart Study: https://framinghamheartstudy.org). The first two sets contain house prices and characteristics jointly. Because of the relatively low sam-

ple sizes we relied on linear regression models as prediction functions, rather than non-parametric models. For the Framingham dataset, we considered extreme gradient boosting as predictors, using a validation sample to apply early stopping, avoiding overfitting in spite of not so large sample size.

Boston Housing dataset [BH]

This data, available in `R` package `mlbench`, contains 506 observations (one per suburb) and 13 variables, amongst which: the median value of owner-occupied homes (the target), crime rate, average number of rooms, pupil-teacher ratio. The complete list is provided in table 9.1. In order for the reward distribution to be normal-shaped, we took score function $score(\tilde{x}, y) = -|y - \hat{m}(\tilde{x})|^{1/2}$.

name	**label**
target	
medv	median value of owner-occupied homes in USD 1000's
features	
crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
nox	nitric oxides concentration (parts per 10 million)
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built prior to 1940
dis	weighted distances to five Boston employment centres
rad	index of accessibility to radial highways
tax	full-value property-tax rate per USD 10,000
ptratio	pupil-teacher ratio by town
b	$1000(B - 0.63)^2$ where B is the proportion of blacks by town
lstat	percentage of lower status of the population

Table 9.1: Boston Housing dataset variable description, obtained from `R` package `mlbench` documentation associated to dataset `BostonHousing`.

House prices dataset [AMES]

The AMES dataset, available at http://jse.amstat.org/v19n3/decock/AmesHousing.txt consists of 2930 observations of house value, which we log-scaled, and 81 characteristics such as overall quality, year of construction, surface information. The set of features was brought down to ten continuous variables, listed in table 9.2. We used a mean squared error in our score function.

Coronary Heat Disease dataset [CHD]

Collected by Boston University through the Framingham Heart Study[1], a dataset related two coronary heart disease is available here: https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset. The dataset has 15 attributes and class label and the patients were observed for 10 years on various risk factors, which we categorize as demographic, behavioral and medical, see table 9.3.

Interestingly, this dataset allows us to slightly extend the framework we presented up to now: (a) we will assume that some initial data, typically demographic, is available (b) there is not necessarily a one-to-one relationship between actions and features. For (a) we will do an experiment where

[1] https://framinghamheartstudy.org

name	label
target	
SalePrice	the property's sale price in dollars
features	
OverallQual	Overall material and finish quality
GrLivArea	Above grade (ground) living area square feet
YearBuilt	Original construction date
GarageCars	Size of garage in car capacity
TotalBsmtSF	Total square feet of basement area
GarageArea	Size of garage in square feet
1stFlrSF	First Floor square feet
FullBath	Full bathrooms above grade
YearRemodAdd	Remodel date
LotArea	Lot size in square feet

Table 9.2: AMES dataset variable description, obtained from Kaggle competition https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data, the original documentation Dean De Cock being available at http://jse.amstat.org/v19n3/decock/DataDocumentation.txt which contains much more details on variables.

name	label
target	
TenYearCHD	10 year risk of coronary heart disease CHD (binary)
demographic	
male	is the patient male
age	age at exam time
education	education level (1-4)
behavior features	
currentSmoker	is currently a smoker
cigsPerDay	number of cigarettes smoked per day
medical features	
BPMeds	is on blood pressure medications
prevalentStroke	previously had a stroke or not
prevalentHyp	prevalent hypertension or not
diabetes	patient has diabetes
totChol	total cholesterol in mg/dL
sysBP	systolic blood pressure in mmHg
diaBP	diastolic blood pressure in mmHg
BMI	body mass index
heartRate	heart rate in beats/min
glucose	glucose level in mg/dL

Table 9.3: CHD dataset variable description obtained through Kaggle data description https://www.kaggle.com/amanajmera1/framingham-heart-study-dataset.

the gender is available and one where all five demographic variables are available at the beginning (action 0). For (b), we will consider a one-to-one relationship, except for systolic and diastolic blood pressure, acquired simultaneously with a blood pressure measurement (max/min values). Such relationship can be nicely represented with a feature-action relationship matrix as illustrated in figure 9.2.

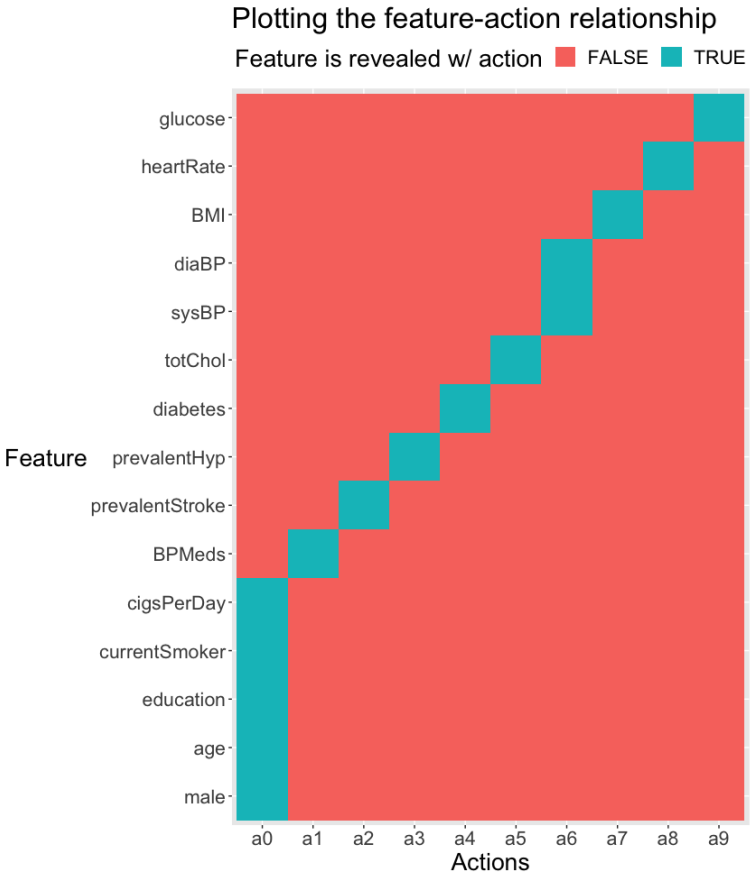

Figure 9.2: Feature-action relationship matrix for CHD dataset: demographic variables are assumed to be available initially, associated to action 0, because those are considered to be zero cost variables. Systolic and diastolic blood pressure are collected through a blood pressure measurement (action 6), all the others are collected with their own specific measurement / question e.g. blood glucose measurement for glucose, heart rate monitoring, asking for diabetes history.

In this case we chose to take the following score function:

$$score(\tilde{x}, y) = -\hat{m}(\tilde{x})(1 - \hat{m}(\tilde{x})) \tag{9.4.1}$$

which is the Gini index or entropy, ranges in [-0.5, 0], 0 indicating we have manage to properly separate the two classes. Note that we do not consider the associated value y but solely the predictor, which supposedly is calibrated on (X, Y) ; supposedly this is a much easier target for our neural network. As prediction function we used gradient boosting with early stopping as regularizer, setting aside a portion of the training set to keep AUC criterion in check.

9.4.3 Global performance

For each problem, we replicate the train/test split at random 10 times, calibrating our model as well as three baselines on the training set and evaluating on the test set. Toy model #1 being a classification problem we used the Area Under the Curve (AUC) metric for comparison, whilst we used the Root Mean Squared Error (RMSE) for all other problems.

In table 9.4 we report the test set performance average and its standard deviation in parenthesis. The approach we propose is labelled SmartQ in the table ; the oracle corresponds to the model using all p features ; best q subset relies on the subset of features which performs best on the training set ; CART q is a CART decision tree with depth q, calibrated using `R` package `Rpart`. On all problems, whether it be toy models or benchmark datasets, SmartQ outperforms both best q subset and the decision tree with maximum depth.

Problem	**Metric**	Bound	Oracle	SmartQ	Best q subset	CART q
Toy models						
#1	AUC	1	1 (0)	0.87 (0.01)	0.75 (0.02)	0.81 (0.01)
#2	RMSE	0.2	0.3 (0.01)	0.37 (0.01)	0.46 (0.01)	0.41 (0.01)
#3	RMSE	$\sqrt{2}$	1.56 (0.02)	1.57 (0.03)	1.83 (0.03)	1.84 (0.02)
Benchmark datasets						
BH	RMSE		4.99 (0.57)	4.92 (0.54)	5.33 (0.54)	5.16 (0.64)
AMES	RMSE		4.3 (0.22)	4.56 (0.14)	4.67 (0.19)	5.62 (0.15)
CHD $[1-5]$	AUC		68.88 (2.01)	69.12 (1.85)	68.77 (1.76)	64.8 (2.89)
CHD [1]	AUC		69.73 (1.92)	62.38 (3.2)	61.04 (4.35)	60 (3.41)

Table 9.4: Average and standard deviation of prediction performance on test sets, on 10 different train/test splits. For the benchmark datasets, we sampled at random the train/test splits every time, whilst for the toy models we simply generated new train/test datasets each time. For the CHD dataset, we considered two versions: the case where variable 1 (male) is known initially (problem label: CHD [1]) and the case where the five first variables, demographic and behavior features, are initially known (problem label: CHD $[1-5]$). On each problem, SmartQ performed better than the best q subset and the CART decision tree, getting close to the oracle predictor. For the toy models we added as indicator the best possible performance for the toy models considered, based on their intrinsic variance.

9.4.4 Variable importance

We take a look at a particular instance of the application on AMES dataset. In this instance the best linear model found (best q subset) relied on variables: *OverallQual*, *GrLivArea*, *YearBuilt*.

In figure 9.3 are represented the question sequences asked by the smart questionnaire on a test set of the AMES problem. The thickness of the arrows indicate the proportions of cases making the transitions. The variable found for the best q subset still matter a lot in the smart questionnaire, but it seems to be more interesting for instance, depending on *OverallQual* observed to ask for *GarageArea* or *YearBuilt*. Subsequently, *GrLivArea* is often asked for, but mostly when *GarageArea* was queried, otherwise many other variables such as *1stFlrSF* and *FullBath* are asked. The questionnaire manages which information is more important to get depending on previously recorded values, as expected.

In figure 9.4 we plot the normalized variable importance from random forest algorithm (red) and alongside the pourcentage of cases, on the test set, where each variable was asked for. In this instance it appears that a positive relationship does exist between the two quantities, albeit not

perfect, which suggests these variable importance scores could be a heuristic for which variables should appear in our smart questionnaire, which could be useful when dimension may come as an issue.

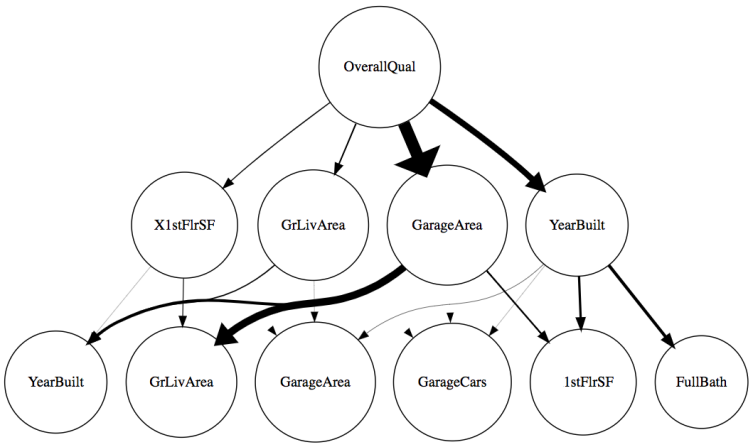

Figure 9.3: Diagram showing the paths taken during the application of the smart questionnaire on the test set on AMES dataset. Note: contrary to a decision tree which explicits the directions, this graph is only a representation of which features where asked, not why.

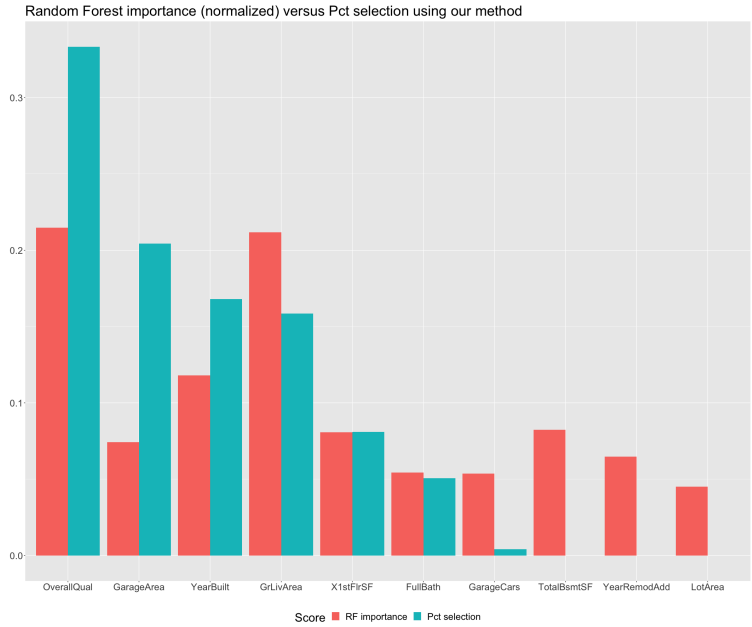

Figure 9.4: AMES, variable importance from global random forest and pourcentage of cases each variable is asked for.

9.4.5 Visualization

The question of how to visualize the smart questionnaire can be quite tricky, unless a large proportion of the features are binary, which helps tremendously. We consider examples on the toy models.

Toy model #1

Because the features are binary, the usual decision tree representation can be applied. The decision tree that serves as baseline is presented in figure 9.5 whilst the questionnaire algorithm we found is presented in figure 9.6.

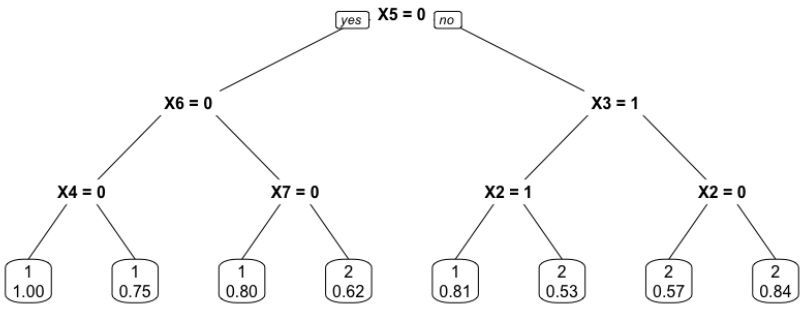

Figure 9.5: Decision tree baseline

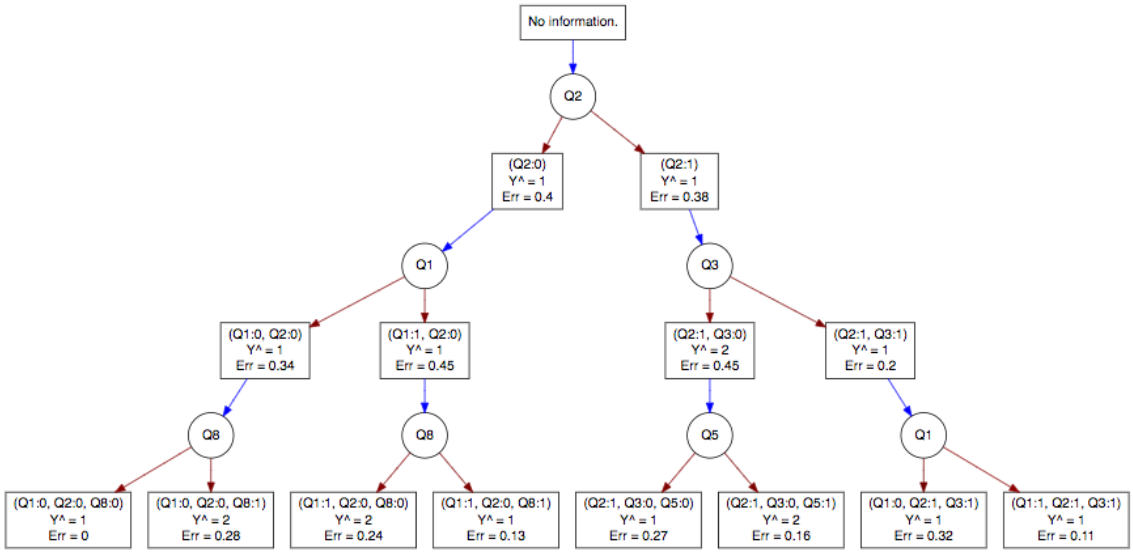

Figure 9.6: Questionnaire: $\hat{Y}$ indicates the prediction assigned to each node, represented by squares here (1/2 coding rather than 0/1 coding) and the error term is the predicted error.

Toy model #2

The intelligent questionnaire found started in the two first layers by asking for binary features, as such the questionnaire can be summarized quite simply, as follows. First, element X_1 was queried. Then element X_2 was asked for when X_1 revealed to be 0, and element X_3 was asked for when X_1

revealed to be 1. Then the final question was:

$$X_1 = 0, X_2 = 0 \to X_4$$
$$X_1 = 0, X_2 = 1 \to X_6$$
$$X_1 = 1, X_3 = 0 \to X_6$$
$$X_1 = 1, X_3 = 1 \to X_5.$$

The decision tree alternative is presented in figure 9.7.

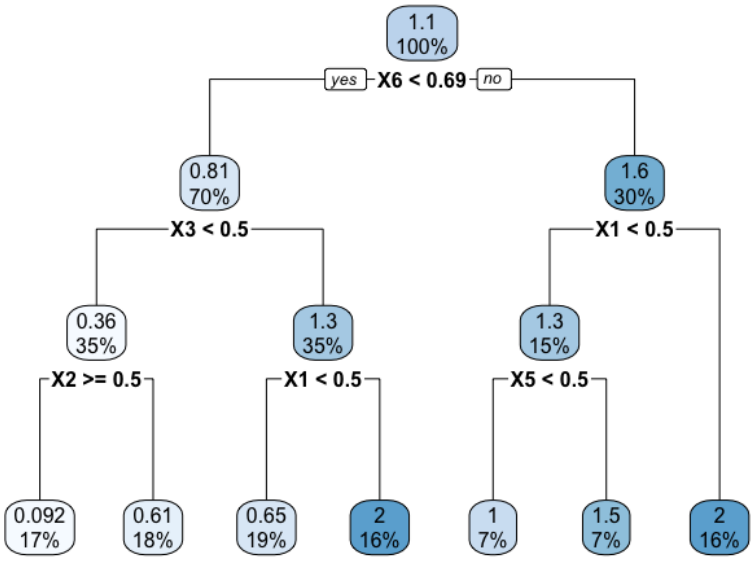

Figure 9.7: Decision tree baseline

General visualization

In the case of toy model #3, we can visualize the questionnaire sequentially: first, question 3 was systematically asked, because we were able to learn the importance of asking for it before any other variable. Then, depending on the value of variable 3, we basically selected either 1 or 2 if $X_3 > 0$ and 4 or 5 otherwise, as shown in figure 9.8. We can observe that in the low extreme we recommended to ask question 7, which is actually an unimportant variable, this is related to poor estimations at the boundaries. Also, because in this setting there are several optimal solutions for the questionnaire, the choice amongst 1/2 and 4/5 can seem quite bizarre. The lines added are found by applying a CART prediction algorithm over which question to ask according to the questionnaire, which can be seen as a surrogate model. Now, focusing for example on the case where the revealed X_3 would belong to interval $[-2, -1]$, we would then proceed to ask question 4, and then, systematically, we asked for question 5, as figure 9.9 illustrates.

Going to the AMES dataset, which is much more interesting we consider an example which starts by asking for *GrLivArea*, the second questions are presented in 9.10, and the third questions are represented in figures 9.11, 9.12 and 9.13, depending on the actual value of *GrLivArea*.

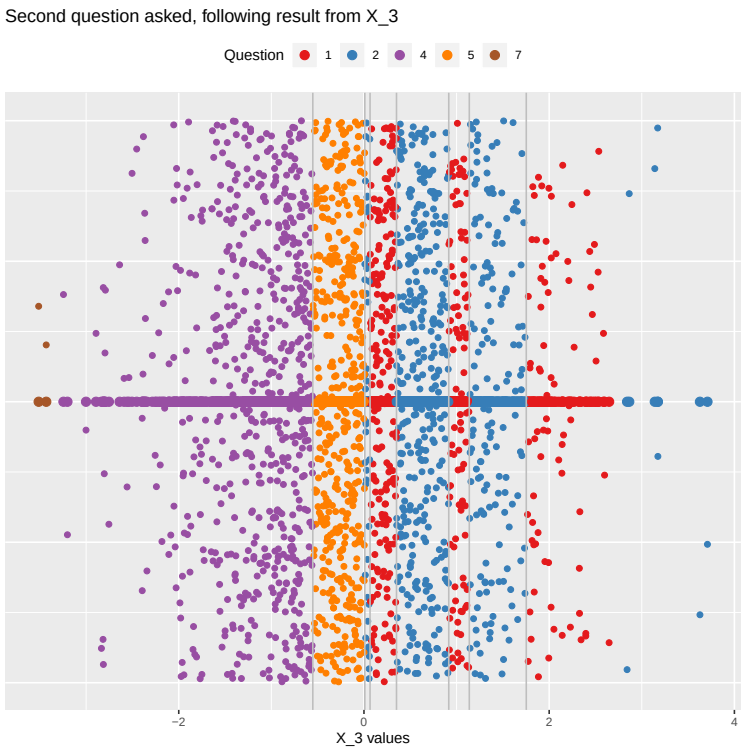

Figure 9.8: Visualizing smart questionnaire decision-making on toy model #3. X_3 is known, what second question should we ask ?

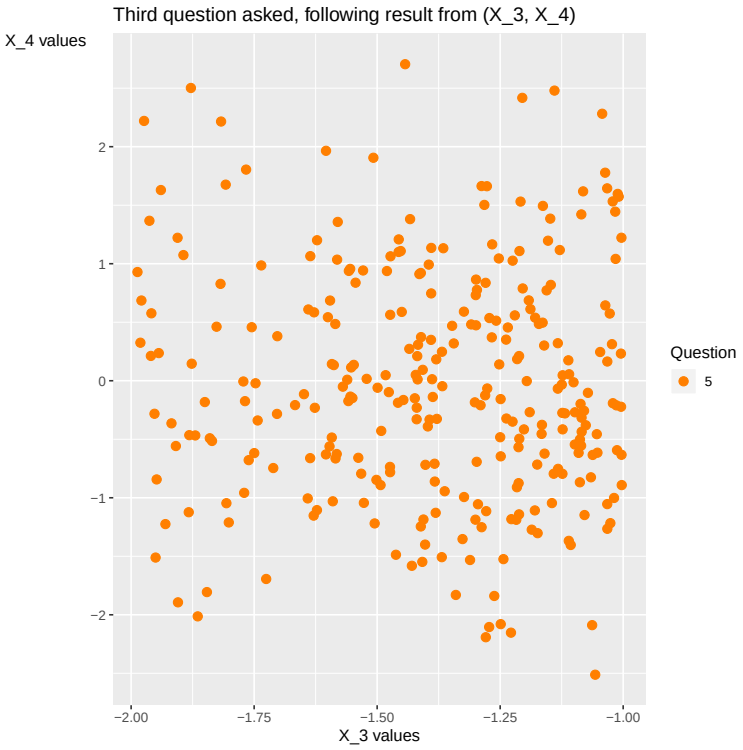

Figure 9.9: Visualizing smart questionnaire decision-making on toy model #3. X_3 is known to be in $[-2, -1]$, we asked for question 4, what third question should we ask ?

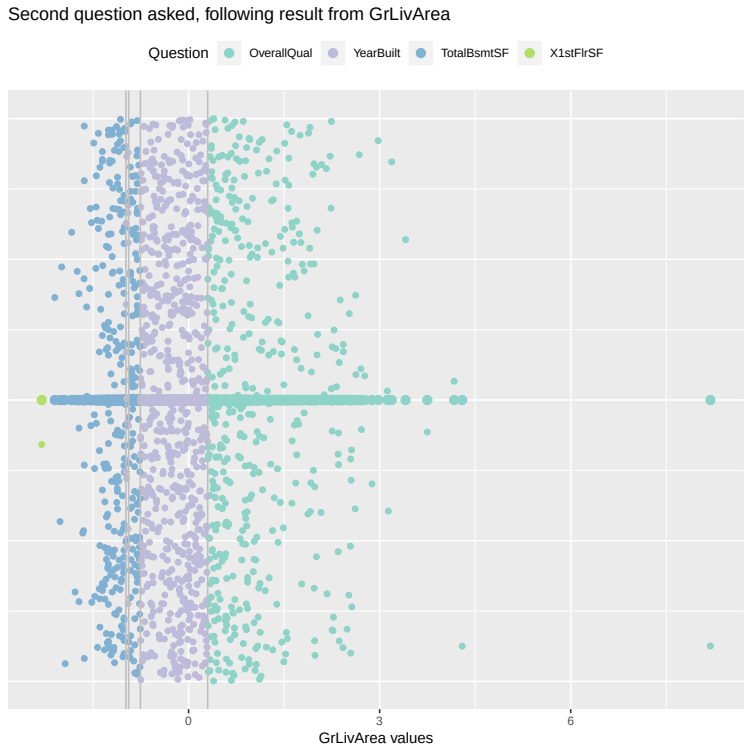

Figure 9.10: Visualizing smart questionnaire decision-making on AMES dataset.

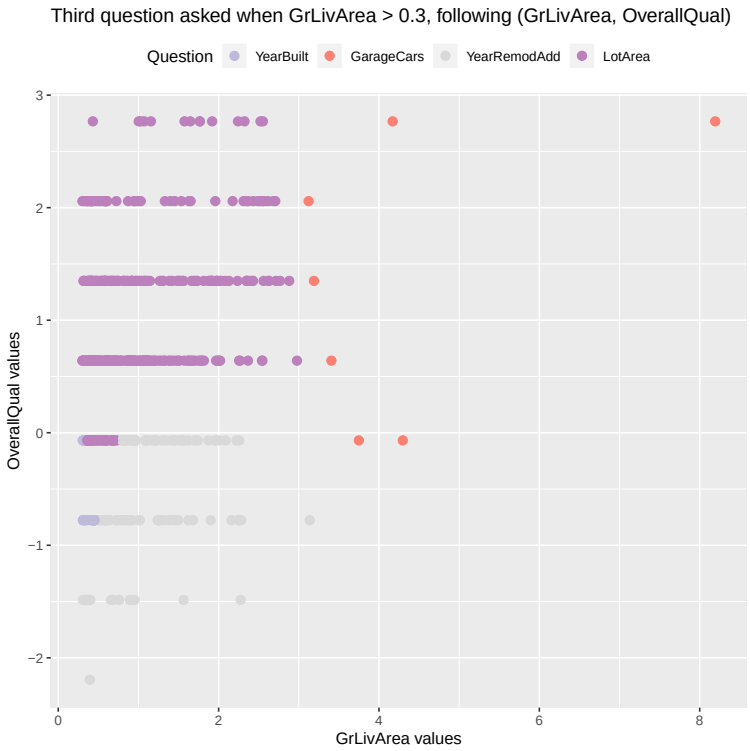

Figure 9.11: Visualizing smart questionnaire decision-making on AMES dataset.

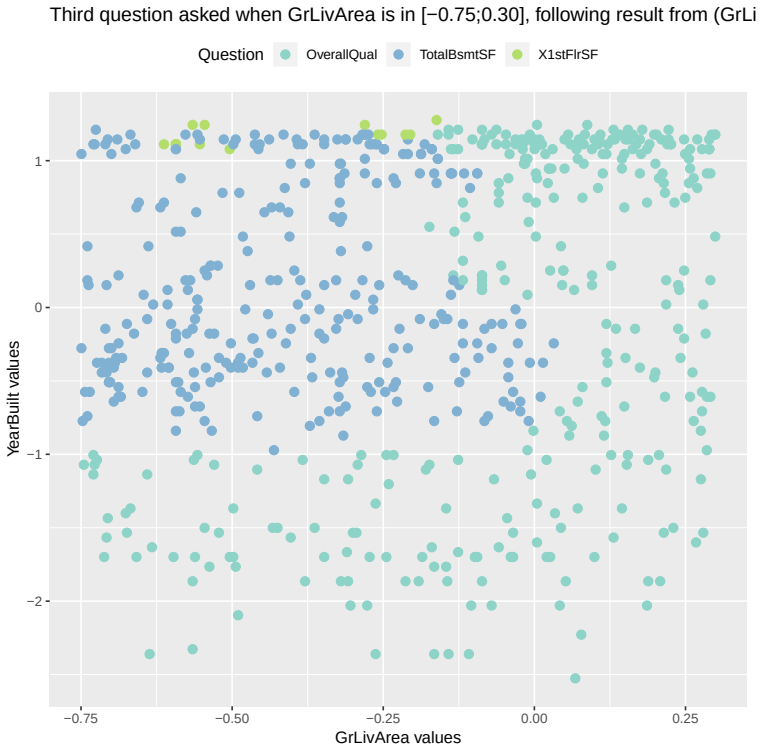

Figure 9.12: Visualizing smart questionnaire decision-making on AMES dataset.

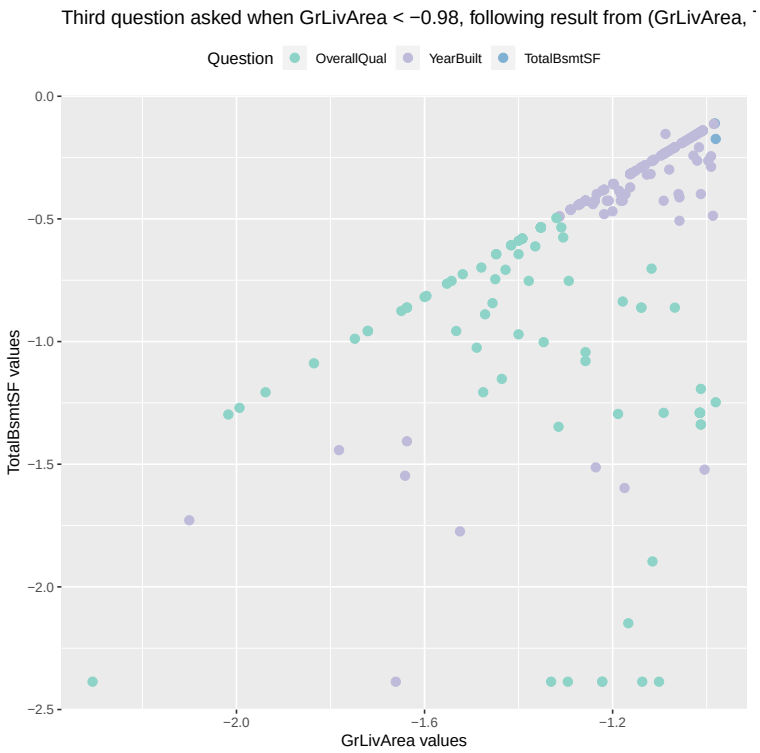

Figure 9.13: Visualizing smart questionnaire decision-making on AMES dataset.

9.5 RL on toy model

9.5.1 Specifics

We considered toy model #3 presented in preceding section: consider $p = 8$ mutually independent random gaussians features

$$X_j \sim \text{Gaussian}(0, 1) \quad \forall j \in \{1, \dots, p\}.$$

The target variable is defined by the simple linear model

$$Y = \begin{cases} X_1 + X_2 + \sigma\varepsilon & \text{if } X_3 < 0 \\ X_4 + X_5 + \sigma\varepsilon & \text{otherwise} \end{cases}$$

where $\varepsilon \sim \text{Gaussian}(0, 1)$ is the noise random variable, independent of X. Parameter σ, which controls the noise level in the model will be set to $\sigma^2 = 0.4$ in order to have a Sound-to-Noise ratio of 5. We consider the problem of gradually sampling data using the questionnaire i.e. gathering only $q = 3$ elements of feature vector X at a time.

We start with 32 samples for each q-question combination and then at each iteration we sample a total of $32 \cdot \binom{p}{q} = 1792$ elements, accordingly with exploration policies. At every iteration, half of the samples are kept for learning function m and half for the neural network Q-value approximation. To approximate m we will rely on random forest algorithm. At each iteration, we will also follow greedily the current questionnaire on a free-test dataset of 2000 samples. This will allow us to track the performance improvements of the smart questionnaire constructed and see which exploration method performs well and under which conditions.

We considered the softmax and ucb exploration policies, presented in the methodology section earlier, setting, quite arbitrarily two different exploration parameters. For all but one of the softmax approaches, we used the square root absolute prediction error instead of the squared error, because the latter has a highly-skewed distribution towards zero, which seems to be causing issues in the neural network training. It however may be that some parameters in the training on the neural network should be changed for the approach to perform properly.

The template neural network used for layers 2 and 1 had the following characteristics: fully-connected neural network, with four hidden layers of 24 neurons, input of size 24 (the first 16 representing the partially-known vector, the last 8 indicating which action is selected) and with output size 1.

In order to obtain our results, we repeated each scenario i.e. exploration policy in 10 experiments with set seed values.

9.5.2 Results

In order to analyze and compare the algorithms performances, we produced a few figures, which can be categorized in two groups, the first providing insight into how well the intelligent questionnaire computed performed on the test samples throughout the online learning process, tracking for instance *final regret* in figure 9.14, checking what the first question selected was, figure 9.15, and if overall the right questions were asked, figure 9.16, which we can very well do because we know the form of the optimal questionnaire. The second group, which is a little larger, essentially focuses

on the behaviour of the different exploration policies and that is: on the acquired samples, in how many cases did we collect a (1,2,3) or (3,4,5) sample, figure 9.18, or a sample with a proper couple i.e. in the set $\{(1,2),(1,3),(2,3),(3,4),(3,5),(4,5)\}$, figure 9.19 and finally the minimum and maximum proportion of triplet sampled in figures 9.20 and 9.21 respectively.

Test results

Let us begin our analysis on focusing on the results from the test sets first. Examining figure 9.14, which represents the RMSE distribution over time steps and for the different policies, we can make several observations: (a) ucb methods take a longer time to reduce RMSE in contrast with softmax exploration, (b) ucb methods also present overall more variability than softmax experiment do, (c) the reward distribution matters, as the softmax policy relying on original squared errors is the only on which seems to stuck to a floor level, two times larger than the lower bound ($\sigma = \sqrt{0.4}$).

One can note that, because of the particular parameters, in the case of softmax the less-exploratory option ($\beta = 12$) is preferable to more exploratory one ($\beta = 6$), but if we would to increase it too much, we would be in much trouble (purely greedy strategy); in the case of UCB, we can see that in this case it was preferable to pick a larger exploration bonus in order to make the solution converge faster.

Looking at which first action was selected from the learnt questionnaire throughout this process, figure 9.15, we see again how the softmax approach allows to converge quickly to picking question 3 first for almost all experiments around step 15, whilst this proportion is around 75% for ucb(100) and 50% for ucb (50), which accounts for the differences in RMSE scores observed previously. The same reasoning can be applied for the softmax policy relying on squared errors, whose training seems compromised after a given number of steps. *Open question: did the data size had something to do with the neural network training ?.*

Akin to the first graph analyzed, figure 9.16 presents the percentage of cases where the right questions were asked i.e. starting by 3, then (1,2) if $3 < 0$, (4,5) otherwise. The behaviour is the inverse as the one observed in the first graph unsurprisingly.

Acquired data results

Let us now focus on how the different exploration policies behaved. Examining plots 9.18 and 9.19 one can observe that only the softmax policies focus their exploration phase on the actual important triplets, and consequently couples: for each softmax policy (aside the one with squared errors), the percentage of relevant couples represent 50% of new samples and linearly increases until the 100% ceiling is reached. The relevant triplets increase for those policies is even more steep. For Ucb exploration policies however, the pertinent triplets are very rarely sampled (their are some outliers i.e. good scenarios) and the pertinent couples are sampled in a very variable manner across our experiments: this is clearly not robust throughout experiments. More explanation on this phenomenon is provided by graphics 9.20 and 9.21 which the present the minimum and maximum proportion of sampled triplet. It appears that for softmax policies, the minimum percentage decreases slowly, until eventually being zero (so the triplet becomes ignored in the sampling process), whilst the maximum percentage gradually increases up to around 50% because those policies tend to finally draw triplet (1,2,3) and (3,4,5). In comparison, ucb methods tend to focus on a few triplets, as in most cases the maximum percentage was largely higher than 50%, which simply constitutes less exploration.

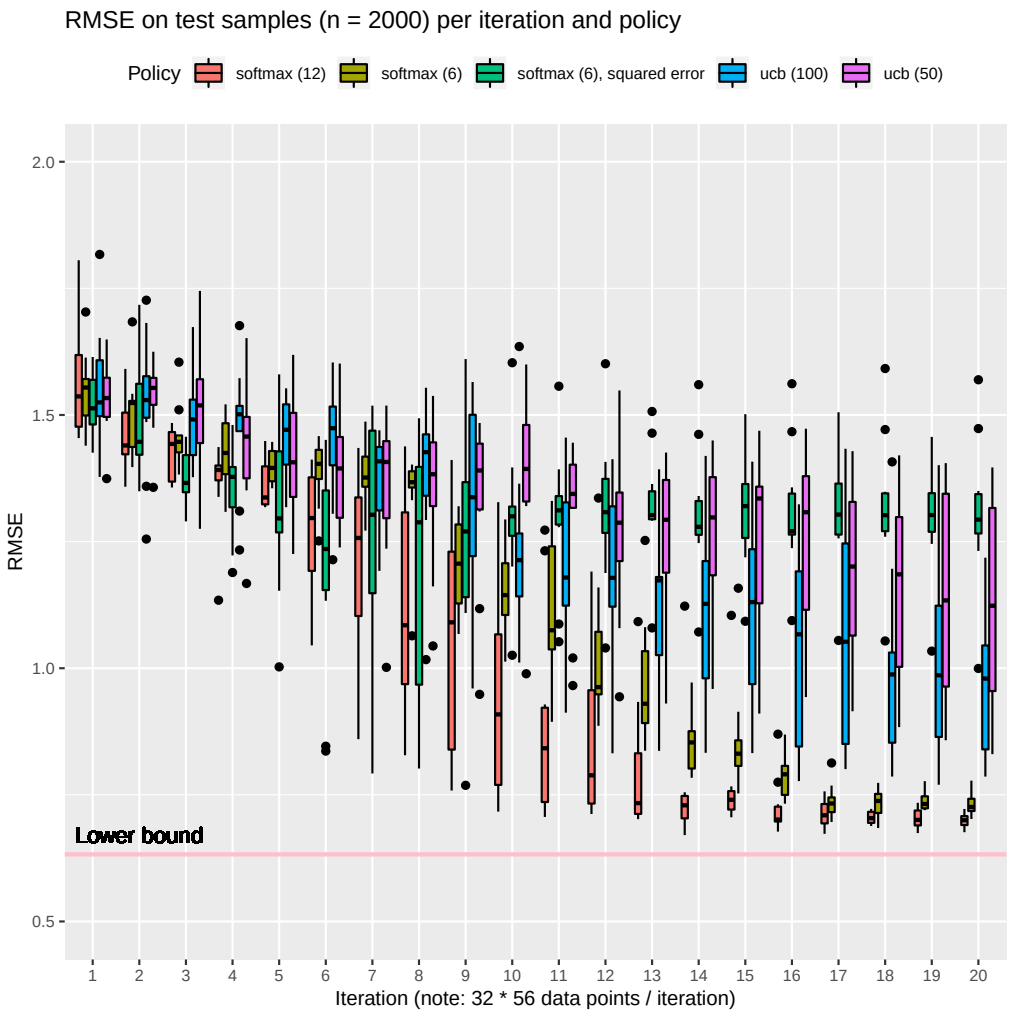

Figure 9.14: RMSE on test samples (n = 2000) per iteration and policy

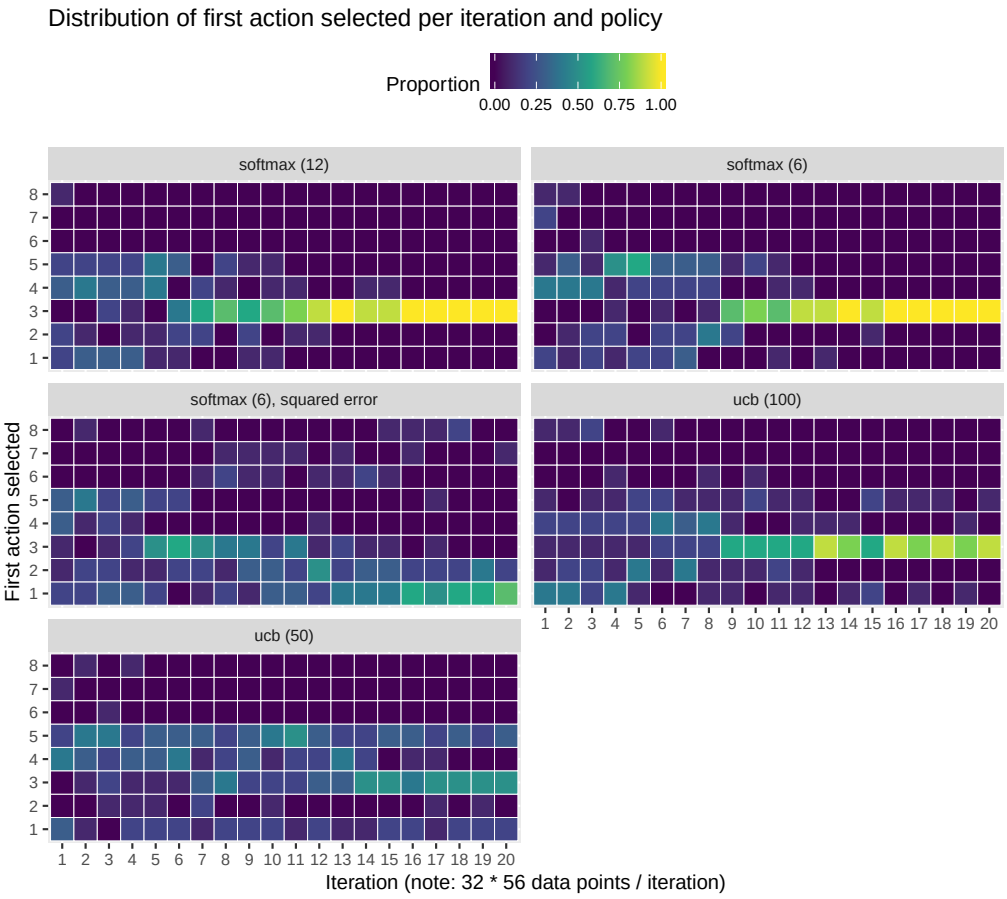

Figure 9.15: Distribution of first action selected, per iteration and policy

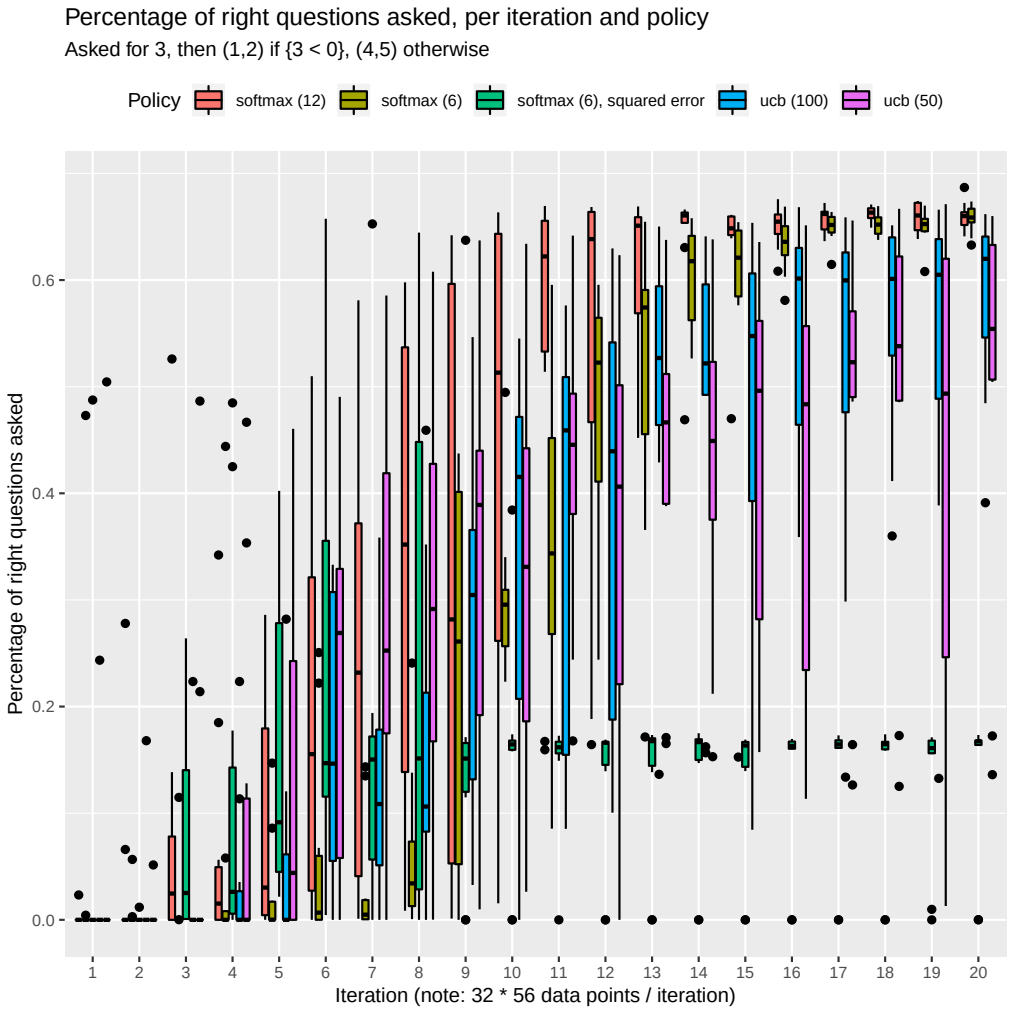

Figure 9.16: Percentage of right questions asked, per iteration and policy

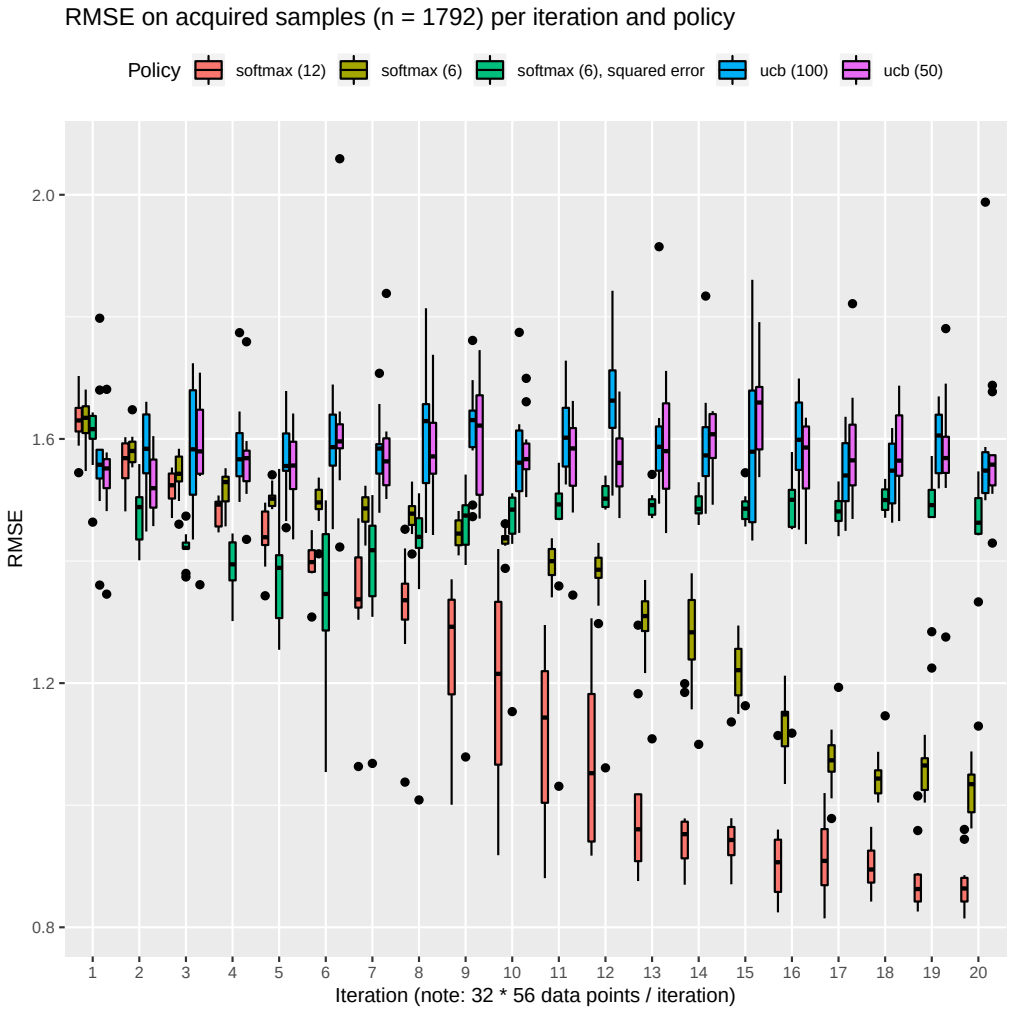

Figure 9.17: RMSE on acquired samples (n = 1792) per iteration and policy

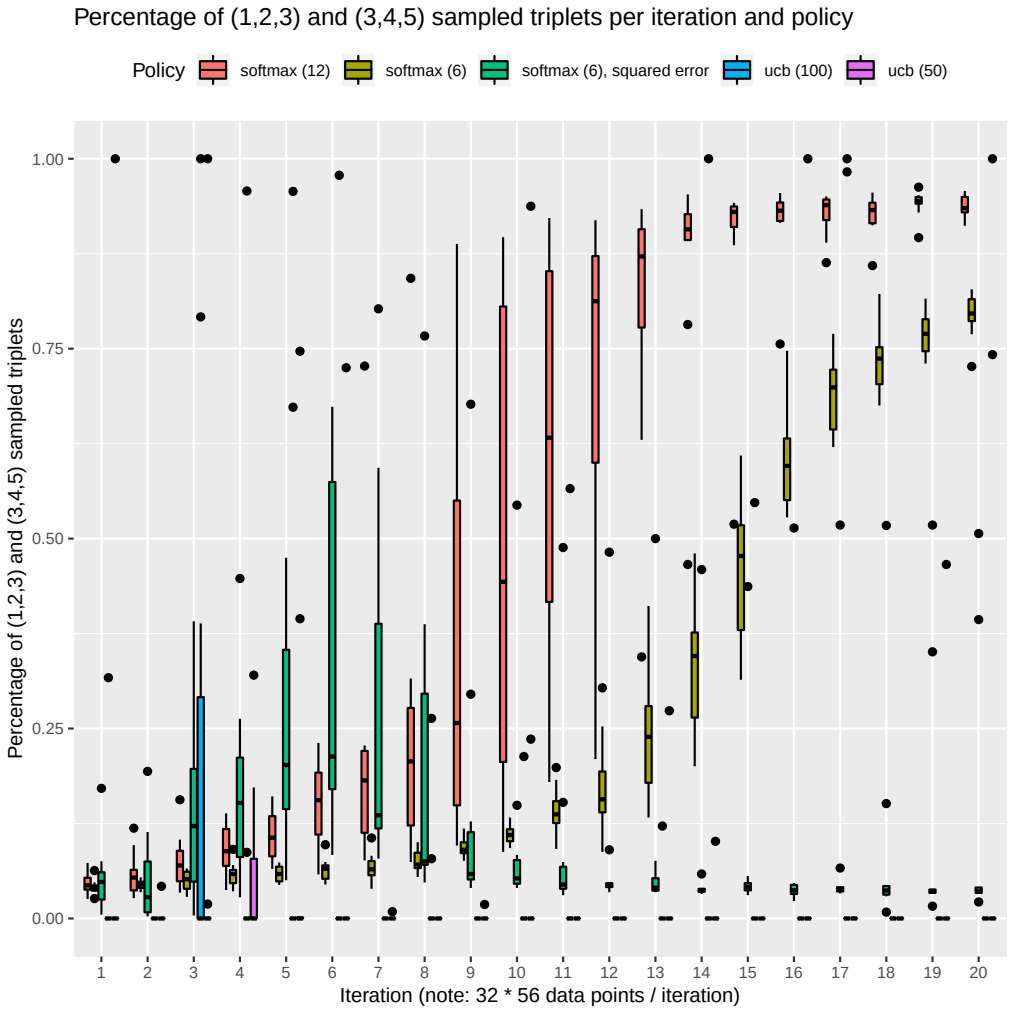

Figure 9.18: Percentage of (1,2,3) and (3,4,5) sampled triplets per iteration and policy

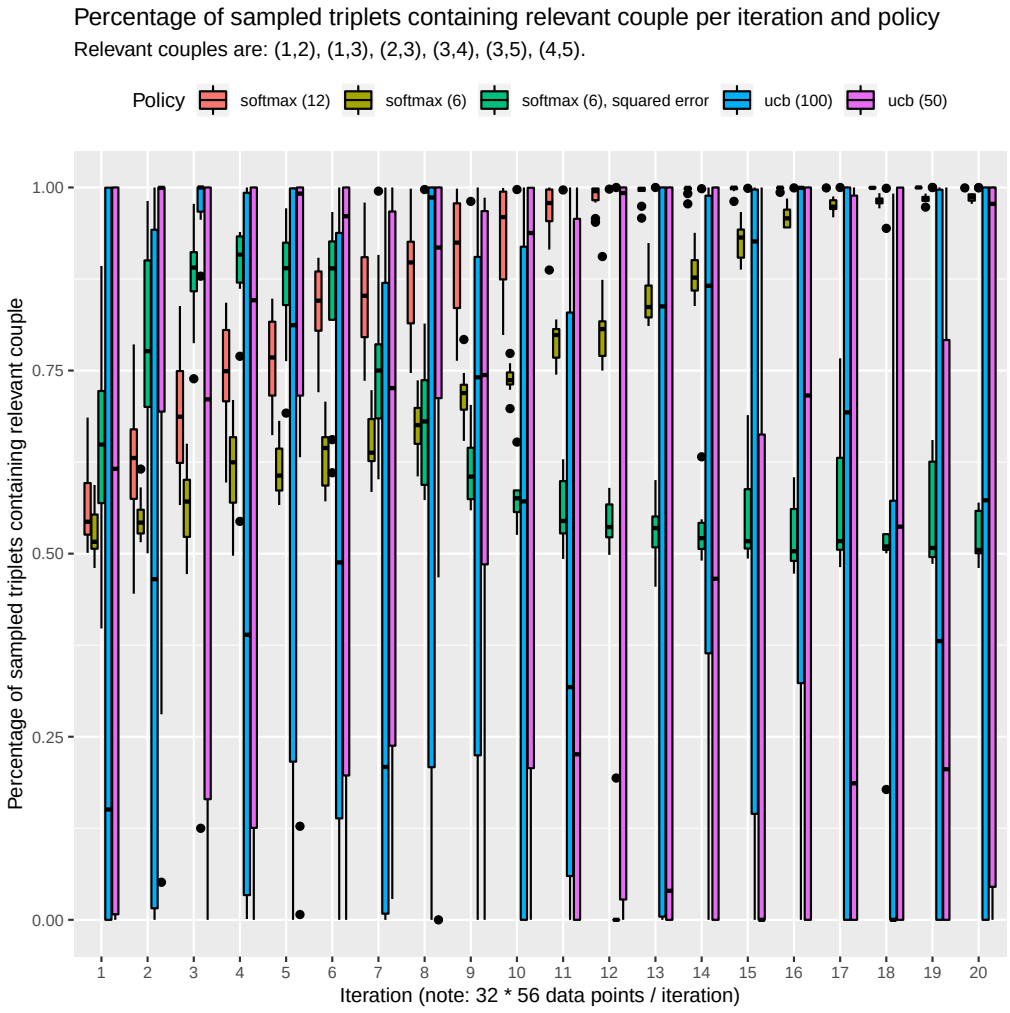

Figure 9.19: Percentage of sampled triplets containing relevant couple per iteration and policy

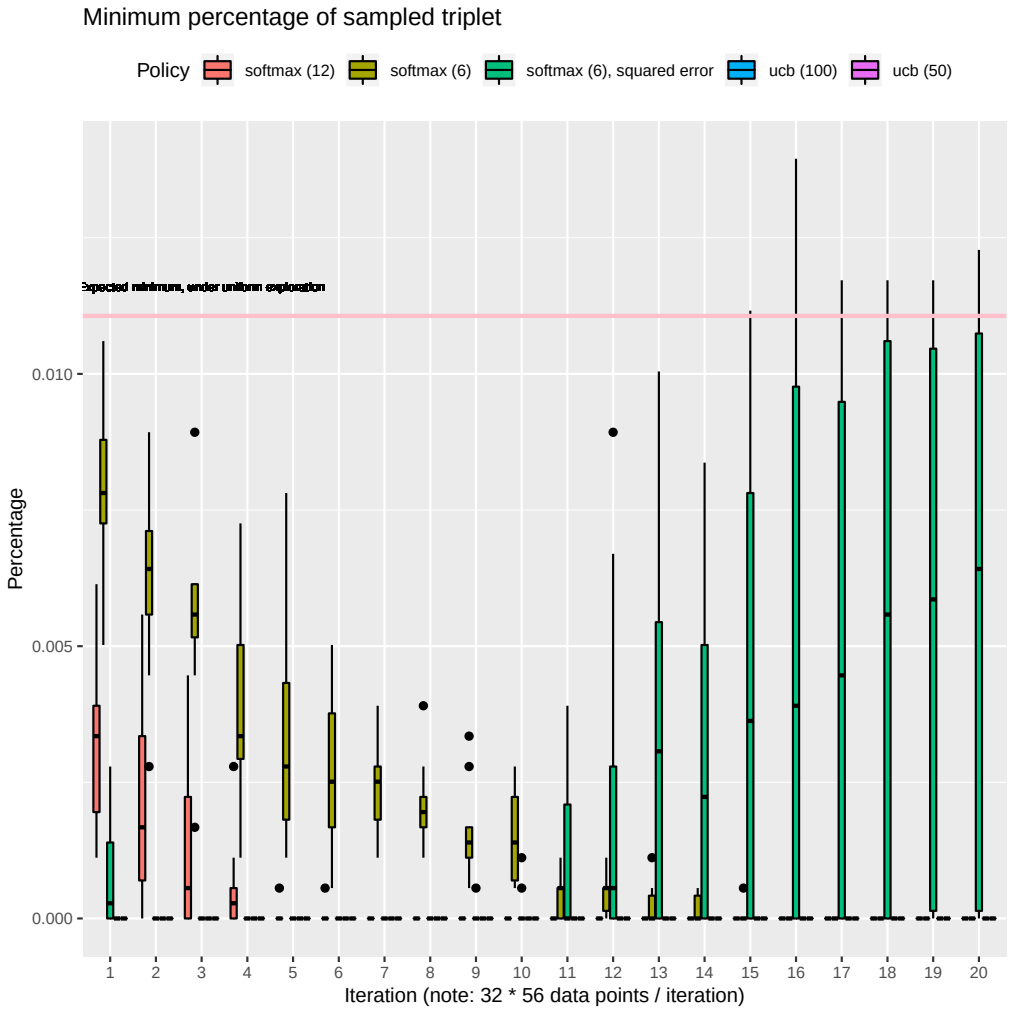

Figure 9.20: Minimum percentage of sampled triplet

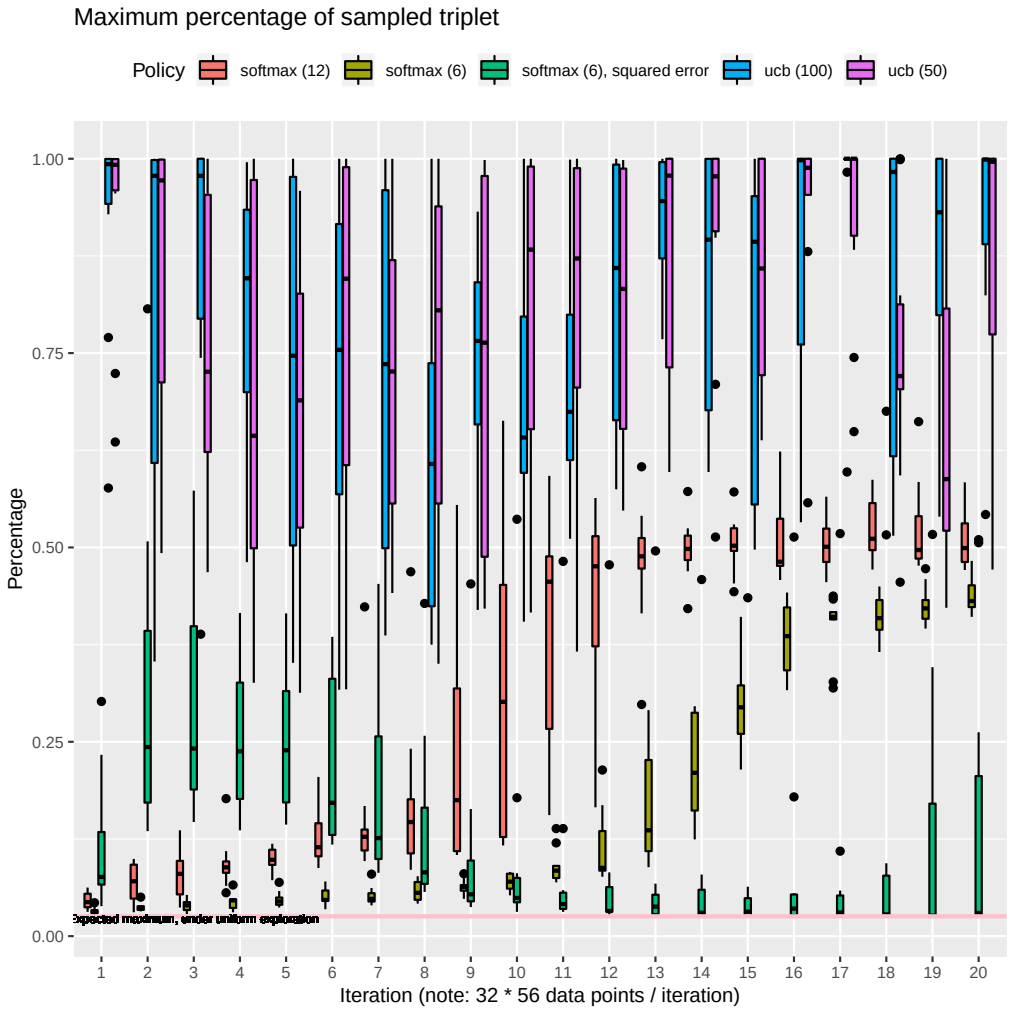

Figure 9.21: Maximum percentage of sampled triplet

9.5.3 Suggestions

In this experiment, it appears that the softmax exploration policy seems to be more easy to deal with compared to ucb from an exploration perspective : it allows for much more exploration, without waiting for next steps ; as such the learning happens more quickly and when we finally differentiate in terms of Q-values it clearly states out, whereas in UCB it is not so clear, performs poorly in comparison in terms of cumulative regret as well as final regret. Also it seems more intuitive to compare Q-values amongst themselves rather than with an exploration bonus of a completely different nature. Also, it seems that reward distribution (at the last layer of the questionnaire) does matter, comparing the case where the squared error was taken instead of the root absolute error.

Something we thoughtfully omitted, is that the reward function changes throughout iterations, since it is defined based on our prediction $\hat{m}$ of m, which is updated as more data is actually available. We are fortunate enough, or in a sufficiently simple setting at least, for this approach to be working. Now, it seems reasonable that we would have some idea of the variance of our prediction, which opens the possibility to learn not the exact error but a lower upper bound on it (optimism-based approach).

Finally, we could consider a more permissive version of the game, where we can actually ask more than three questions, but are penalized for it. That way, we could potentially learn much more, and it would open the way to more diverse exploration strategies.

9.6 Conclusion

In this work we built an adaptive predictive-questionnaire under constraint over the number of questions, motivated by the important balance between data acquisition and user experience.

As this is a sequential decision-making problem we used a Markov Decision Process to model it. Furthermore, its episodic and hierarchical structure allowed us to apply an approximate dynamic programming approach to learn the best adaptive questionnaire based on available data on couple (features, target), which is the standard setting in supervised learning. Finally, we have shown on three toy models as well as two classic benchmark datasets that our approach outperforms (a) a decision tree submitted to the same budget constraint of questions (b) classic models based on the most informative subset of questions. Option (b) being non adaptive and option (a) being limited to one-dimensional splits of data, our approach allows for much more flexibility. We have also showed that this approach works online, trying classic exploration policies for one of the toy models, which was not evident it would as the reward where are trying to learn is data-dependent. As a continuation of this work we have a few ideas for further research.

Stopping criterion

In the algorithm constructed, we assumed a budget constraint over the features requested. This approach makes sense in some applications as it reduces globally the number of requests from us to the user and it simplifies some computational aspects.

We could also consider the case where we would stop asking questions as soon as we believe we

have gathered enough information on X in order to predict Y to a satisfying level:

$$stop(\tilde{x}) = \begin{cases} \texttt{true} & \text{if } \hat{\mathbb{E}}_{(X,Y)|\{\tilde{X}=\tilde{x}\}}\left[score(\tilde{X}^{final}, Y)\right] \geq \tau \\ \texttt{false} & \text{otherwise} \end{cases} \tag{9.6.1}$$

writing $\tilde{X}^{final}$ the terminal state reached by the algorithm. This criterion pre-supposes that we would construct an estimate of the expected score given the information currently available. Although it might complicate the resolution method, it could be of more interest for practical use.

Scaling with (p, q)

In our approach we relied on an approximation of state-value functions based on neural networks, without considering much on the dimension parameters. It is apparent that the higher p and/or q, the more difficult the problem in a blunt overall search. The exploration approaches from active & reinforcement learning will surely be handy to help identify q-sized subsets which are uninformative and handle this potential dimension issue. To set some ideas, we represent in figure 9.22 the dimensions in the binary features setting for a given tree depth q and feature vector size p: very quickly the dimension explodes with p, unless q is systematically brought down to a very low number to compensate.

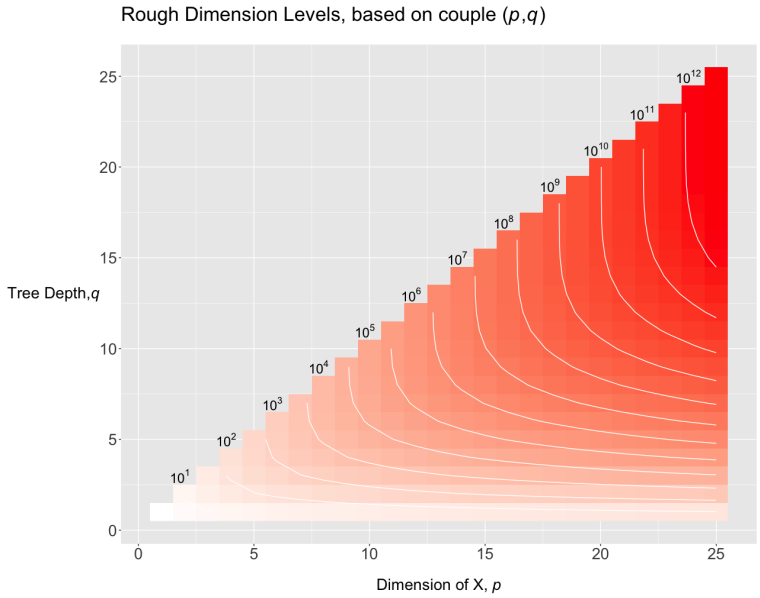

Figure 9.22: Number of states $N(p, q)$, assuming binary features, as a function of p, the dimension of X and q allowed tree depth.

Truthful responses & more uncertainty

Throughout this work we assumed that when an element of X is asked for it is revealed exactly. As such, the same element is never asked for twice and we completely trust the response given. In some applications, the order and overall position of a question in a survey affects the answer given, which can sometimes have very important consequences, see Magelssen et al. [2016]. Therefore, the answer given should be treated as a noisy version of the truth, where the noise actually depends

on the order followed and previous answers that were given. This would lead us to model the interaction through a Partially Observable MDP Kaelbling et al. [1998] $\mathcal{M}(\mathcal{X} \times \mathcal{Y}, \tilde{\mathcal{X}}, \mathcal{A}, \mathbf{T}, \mathbf{R})$ where

- $\mathcal{X} \times \mathcal{Y}$ is the state space, where couple (X, Y) lives, which are partially known
- $\tilde{\mathcal{X}}$ is the observable state space, where partially known random variable $\tilde{X}$ lives
- $\mathcal{A} = \{1, \dots, p\}$ is the action space, the indexes of feature elements we can collect
- $\mathbf{T}$ and $\mathbf{R}$ are the transition and reward functions.

This representation, with simple extensions, could allow for noise in the observation of X which could be important if the assumption that question order matters does not hold. It is known that in several usecases, question order can matter, see https://www.surveymonkey.com/curiosity/question-order-matters/. The phenomenon is called "priming" and implies that the responses to the questions gotten will be dependent on the order chosen. This is something which is not handled here and could be quite difficult to handle.

Different data acquisition setting

In the planning experiments, we considered to have a complete dataset of couples (X, Y), sufficiently large to propose a proper solution. In the RL section, we considered that the data acquisition part was constrained by the intended application: asking only q questions, only that in some cases the individuals were nice/incented/paid enough to provide us with the target variable to train. Several extensions could be considered, for example: rather than stopping at q questions online, we could consider starting with all p questions, and gradually, building our estimates, starting to progressively but surely lower the number of questions to ask, making sure our certainty level is high enough. An interesting reference on the topic is Provost et al. [2007].

Facilitating online setting

On the online setting we have several ideas to explore, especially when the dimension will become problematic.

Instead of recalibrating globally the predictors, the idea would be to rely on fully randomized trees, which are built progressively and only the associated predictions are recomputed as new data is brought in. With a proper implementation, this would integrate quite nicely with the setting.

Also, because we need to find the best action to take, we suggest that for at least the last layer but potentially for the prior ones, we not learn the expected reward but actually two elements: (a) the expected reward for an action compared to the optimal and (b) the expected reward of the oracle. *Open question: are the two equivalent ?* Of course this is only possible in full information, so in the planning approach.

Communication

This work was partly presented at the UCAI'20 (User-Centered Artificial Intelligence) workshop, in September 2020. This workshop aimed at bringing together people from the HCI (Human-Computer Interface) community and the Artificial Intelligence community.

Part III

Exploration for bandits and MDPs

Introduction

For any Markov Decision Process, if we fully know the transition kernel and reward function, then it is relatively straightforward (assuming unlimited computing power) to find an optimal policy and at least be able to rank policies amongst each other.

In the case where model specifications are learnt based on data (model-based), uncertainty on our estimates must be dealt with appropriately, for example by acquiring more data on highly uncertain regions (exploration). In a model-free approach, we still use data to approximate state value or state-action value functions, hence uncertainty on our estimates must be properly taken into consideration and so we need to explore to properly cover the state and action spaces.

Exploration is supposed to help us improve our knowledge of this environment in which the system evolves, and it is considered to be done efficiently when only promising areas are explored hence the exploration-exploitation tradeoff / dilemma.

To explore it is common to make decisions in a non-greedy manner, for instance: on some epochs we pick actions at random (ε-greedy), we pick the actions with the highest upper confidence bound (optimism, UCB, UCT, UCRL), we draw predicted returns from a stochastic belief and pick the highest (bayesian). In all those cases, we look at actions estimated sub-optimal, often preferring the ones with high reward estimate. See Lattimore and Szepesvári [2020] or Sutton and Barto [2018] for more details on the different strategies, in the different contexts.

Here, we present some special instances of problems which have intrinsic stochasticity in their dynamics, such that it may strongly help in learning rapidly without taking too much consideration on the exploration issue.

The first problem, explored in Section 10 is a multi-armed bandit problem where at each time instance, a random proportion of the bandits is unavailable to play. This leads to the same effect as the ε-greedy policy, where we select an arm at random amongst suboptimal arms with probability ε.

The second problem, studied in Section 11, is an MDP with as state information the position of an object on a bounded segment, the reward function depends on the object position and is deterministic. The action authorized is to move the object of maximum step ϵ, either left or right. The next position is made stochastic with a simple gaussian perturbation, such that for a high enough noise, the action decided is negligeable and we actually don't control the system. There is, in between, a sweet spot such that there is sufficient noise such that we automatically explore the segment, whatever the action selected, but not so much that the system can't even be controlled.

The third and last problem, analyzed in Section 12, is a contextual bandit where the context is drawn from a fixed distribution at each epoch and the reward is a linear function of the context. Assuming that the context is diverse enough throughout time, estimating the coefficients involved in the reward function will go well whatever the behaviour policy. In this particular case we can

specifically show that the regret has the right order with respect to the number of times the game is played and with respect to the dimension of the features. Interestingly, a specific quantity associated to the diversity assumption appears in the regret.

Chapter 10

Multi-Armed Bandit with constrained action space

Summary

The Multi-Armed Bandit problem is a classic problem where an agent has to decide between a fixed set of actions in order to optimize cumulative regret. In this chapter, we study a small variation of the problem, where at each decision epoch one may only select from a handful of the actions. An interesting consequence of this constraint is that we are forced to explore new actions, and therefore leads to proper learning and hence good regret behavior, even when our policy follows a greedy strategy.

10.1 Context

In this chapter we consider an extension of the multi-armed bandit problem, as presented in algorithm 13 below. During T time steps, we have to choose one arm amongst a handful and when we play an arm we collect a reward associated to that arm. Supposedly, some arms have higher rewards than others in average and so we would like to identify such arms and play them as fast as possible. To do so, it is known that one must carefully try all the arms until statistical guarantees are presented to ensure that they may be discarded. In the setting we study here, we assume that at each time step, not all the arms are available to be selected: only Q out of K of the arms are available.

Algorithm 13: Multi-Armed Bandit problem with constrained action space

initialize sum of rewards per arm $s = (0 \cdots 0)$
initialize number of samples per arm $n = (0 \cdots 0)$
input function *policy* which proposes an action based on prior rewards observed and authorized actions
note : $sample\_from$ function embodies the say the system decides at a given time which Q arms out of K are available
note : *reward* function maps to an arm an observation of reward. If rewards where deterministic, then there would be no uncertainty as to which arm is the best, the agent would need only to play them once and the problem would be settled
for $t = 1, \ldots, T$ **do**
    `/* environment specifies authorized list of arms */`
    $\mathcal{A}_t \leftarrow sample\_from(set = \{1, \ldots, K\}, size = Q)$
    `/* user chooses an action */`
    $A_t \sim policy(s, n, \mathcal{A}_t)$
    `/* observe reward */`
    $R_t \sim reward(A_t)$
    `/* update estimates */`
    $s_{A_t} \leftarrow s_{A_t} + R_t$
    $n_{A_t} \leftarrow n_{A_t} + 1$
end
return $(A_t, R_t)_{t=0}^{T}$

From there, you will necessarily explore: as illustrated in figure 10.1 for different couples (K, Q), you will pick with non-zero probability an arm presumed to be sub-optimal according to current estimates.

This inherent stochasticity of the system has consequences similar to taking an ε-greedy strategy, as with some probability we will not choose the estimated optimal arm. Now, there are two majors distinctions to be made: first of all, here we assume that the system has intrinsic stochasticity which basically constrains us to take some actions we would not otherwise, whilst the ε-greedy strategy is a specific exploration-exploitation handling policy, and so pertains to the decision-making of the user. Second major difference, although with some probability we do not choose the estimated best arm, it remains to specify how we choose an arm amongst the the ones which are available. If we play a greedy strategy for example, we will pick the arm which seems to be the best amongst the ones remaining, and so depends on the quality of our current estimates, so in a sense the exploration in that case would still be guided by our estimates, whilst adopting an ε-greedy strategy leads to uniform random exploration.

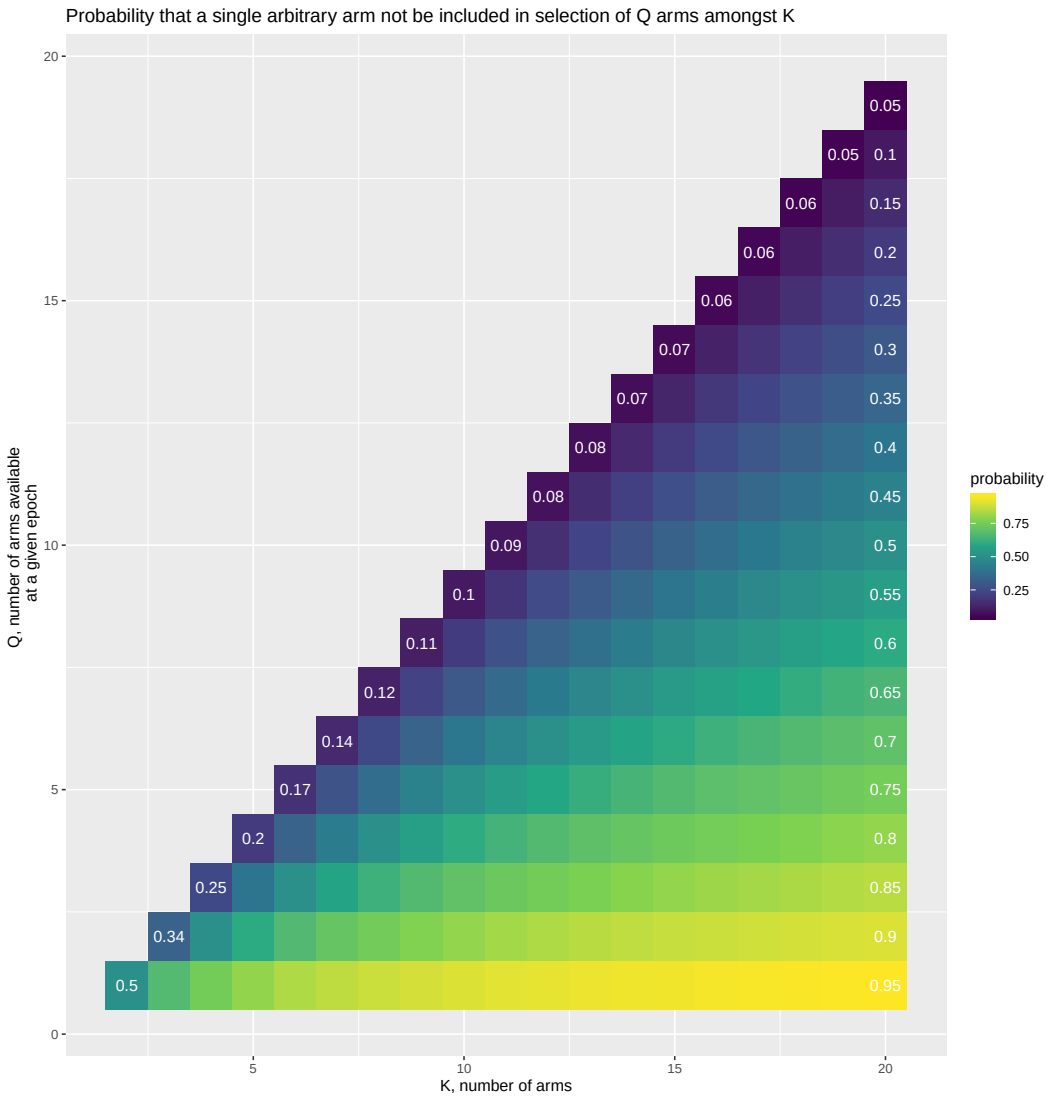

Figure 10.1: Probability of an arm being unavailable when a subset of Q arms out of K are picked.

10.2 Simulations

Let us instanciate this problem as follows: we consider that the game is played during $T = 1000$ iterations, there is a total of $K = 11$ arms, with reward function

$$\forall a \in \{1, \dots, K\} \quad reward(a) = \text{Gaussian}\left(\frac{a-1}{K-1}, 0.5\right) \tag{10.2.1}$$

such that the best-arm-sequence (considering average reward obtained) is Arm K, Arm $K-1, \dots,$ Arm 1.

We consider that at each time step $Q = 5$ arms out of K are available, selected uniformly at random. As a comparative baseline, we look at the same game with a ε-greedy procedure with $\varepsilon = 0.545$ which is the probability that the best estimated action would be unavailable.

The game was played 50 times overall.

What is interesting to witness is the behavior of the learnt policies and how when the problem is inherently stochastic (and so we are restrained to some actions) we learn equally well what the good actions are, see figure 10.2. Note also the comparison with the greedy decision-making, which fails half of the time to identify the best arm.

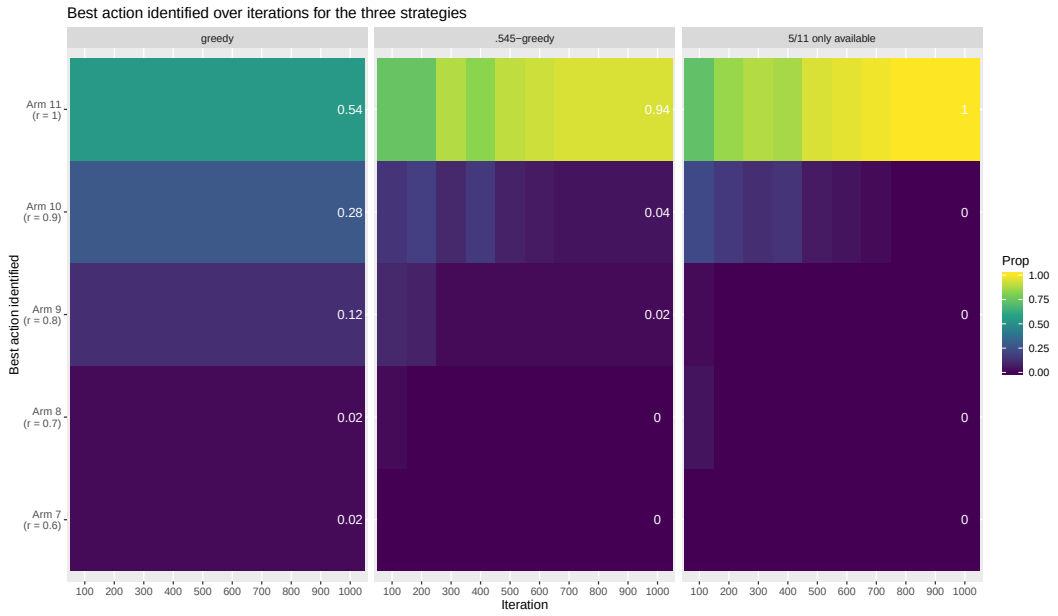

Figure 10.2: Best action estimated. Example: after 1000 iterations, the .545-greedy strategy is able to identify arm 11 as the optimal one in 94% of the simulations.

In figure 10.3 we plot the cumulative reward and regret for each policy, with the cumulative regret up to time t being defined as:

$$\begin{aligned}\sum_{t=1}^{T} \max_{a \in \mathcal{A}_t} \mathbb{E}[reward(a)] - \mathbb{E}[reward(A_t)] &= \sum_{t=1}^{T} \max_{a \in \mathcal{A}_t} \left(\frac{a-1}{K-1}\right) - \left(\frac{A_t - 1}{K-1}\right) \\ &= \frac{1}{K-1} \sum_{t=1}^{T} \max_{a \in \mathcal{A}_t} a - A_t \end{aligned} \tag{10.2.2}$$

with $\mathcal{A}_t$ the authorized set of arms at time t.

Each line in figure 10.3 represents one simulation (so there are 50 lines per color) and one bold line for each scenario is the smoothed average over time of each quantity, to compare easily average behaviours.

One can note the classic unhomogenous behaviour associated to the greedy policy: depending on the first estimates we can be strongly mislead to follow sub-optimal arms. On the other hand the ε-greedy strategy and the constrained setting display a controlled average regret.

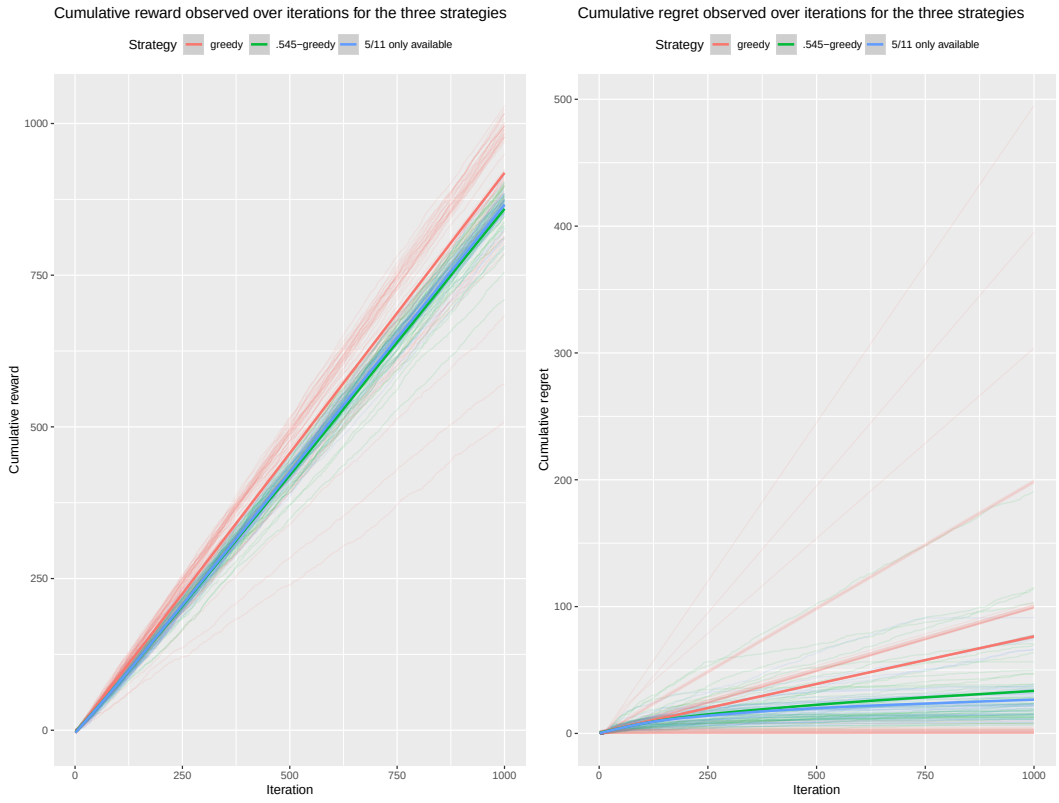

Figure 10.3: Cumulative reward (left) and regret (right) observed for the three strategies. The averages over simulations are the bold lines.

10.3 Conclusion

In this section, we have introduced a variant of the classic Multi-Armed Bandit problem, where at each epoch the allowed action (equivalently decision) space is randomly constrained by the system itself. This leads to natural exploration (even when following the greedy rule on the available decision space) as it does in ε-greedy since with a certain probability the estimated optimal arm is unavailable. Note however that it changes the definition of the regret as the oracle is not the same. Experimentally, we have seen that exploration really does occur, and it shows on the cumulative regret behavior. Finally, please note that if the environment is adversarial and not stochastic, we could be in trouble when following such greedy decision-making.

Chapter 11

Object on a rule

Summary

The exploration-exploitation tradeoff is quite important in Reinforcement Learning as it conditions the knowledge we receive on the system we are interacting with, or controlling depending on the application. Yet, we could conceive of a system with dynamics random enough that we would not need to specify an exploration strategy, and following a greedy approach would be sufficient. Evidently, the dynamics must still be influenced somehow by the agent's actions (our actions), otherwise there is simply no problem to solve. We investigate such a system in this chapter.

11.1 Context

Consider an object with initial position 0.5 on segment [0;1] which may be moved in either direction by a step of maximum range $\epsilon = 0.1$ at every time step. At each step, a reward is perceived depending on the state reached, accordingly with the reward function

$$\mathbf{R} \ : \ [0;1] \rightarrow [0;2.5] \tag{11.1.1}$$

and plotted in figure 11.1.

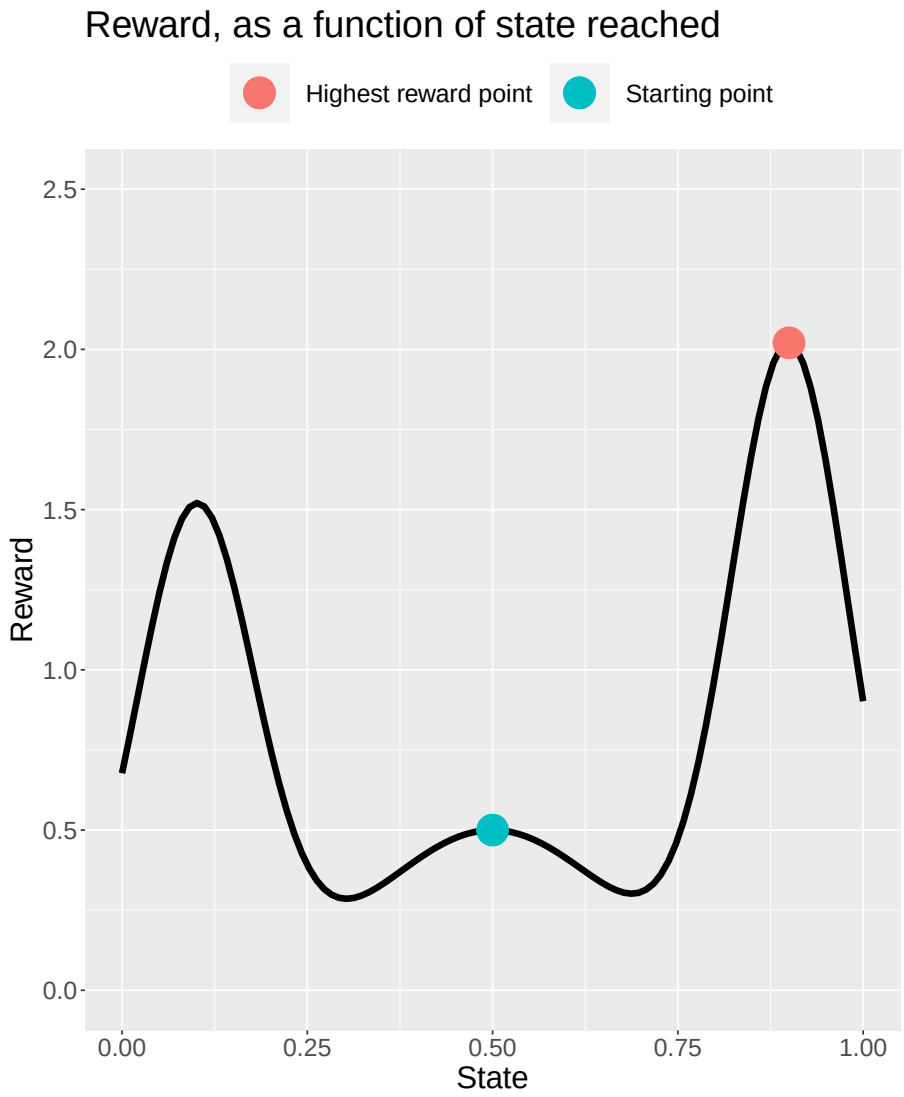

Figure 11.1: Reward obtained, deterministic function of the state reached. The object's initial position is 0.5. The best position to reach is 0.9.

If the object's moves were completely determined by the agent's decisions, then the problem would be to properly evaluate the reward function on the segment [0;1], for it is unknown to the agent at the beginning.

Now, in this section, we are interested to see what happens when the object's next position is influenced by the agent's decisions, but partly stochastic.

We write S_t the object position at time t, $t \in \mathbb{N}$; recall $S_0 = 0.5$, and let A_t denote the agent's decision at time t, based on S_t. We consider the following transition rule:

$$\forall t \in \mathbb{N} \qquad S_{t+1} = \max\{\min\{S_t + A_t + \varepsilon_t, 1\}, 0\}, \quad \varepsilon_t \sim \text{Gaussian}(0, \sigma), \tag{11.1.2}$$

with σ, $\sigma \in \mathbb{R}_+$, is the level of displacement perturbation.

With $\sigma = 0.2$ for example, even trying to keep the object in the center position can lead with high probability anywhere in the segment $[0.1, 0.9]$ in just a few steps. This intrinsic stochasticity is an advantage that must be exploited.

Note that, although we do not explicitly frame it as such, this process is indeed a Markov Decision Process.

We consider the following greedy policy where we go in the direction of the state which is known to us to yield the highest reward: after t steps, we have:

$$\forall s \in [0; 1] \qquad \pi_t(s) = \max\{\min\{(\arg \max_{i \in \{0,\dots,t\}} \mathbf{R}(S_i)) - s; +\epsilon\}; -\epsilon\}. \tag{11.1.3}$$

11.2 Simulations

To illustrate this, we generated 100 games of length $T = 50$ with different values of σ following policy expressed in equation 11.1.3. The values of σ range from 0 to 0.4 with a step of 0.025, so 17 different values.

In figure 11.2, we plotted the couples of state reached and reward observed for different values of σ and it appears that for $\sigma \geq 0.1$ the reward function is properly explored. The view of this particular figure overlays the 100 simulations, so we propose an alternative illustration in figure 11.3 where we plot the highest reward observed for our different runs. One can witness three levels where the policy might get stuck, which correspond to the three modes of the reward functions at positions 0.1, 0.5 and 0.9 and associated rewards 1.5, 0.5, 2 respectively.

In figure 11.4 we plot the cumulative average of rewards obtained throughout time for the different levels of σ, defined as follows:

$$\forall t \in \mathbb{N}_* \qquad \frac{1}{t} \sum_{i=1}^{t} \mathbf{R}(S_i). \tag{11.2.1}$$

One can witness that by increasing σ there is a phase where only some runs manage to reach higher reward levels and progressively the majority become better at it, which is coherent with what we observed on the preceding illustration. Interestingly, the overall average cumulative reward seems to reach its peak for σ around value 0.15. To gain more insight, we focus on the last step cumulated rewards, illustrated in figure 11.5. To this boxplot visualization, we added the average cumulative reward under the *Oracle* policy:

$$\forall s \in [0; 1] \qquad \pi^{oracle}(s) = \max\{\min\{0.9 - s; +\epsilon\}; -\epsilon\} \tag{11.2.2}$$

and under the *Keeping centered* policy:

$$\forall s \in [0; 1] \qquad \pi^{centered}(s) = \max\{\min\{0.5 - s; +\epsilon\}; -\epsilon\} \tag{11.2.3}$$

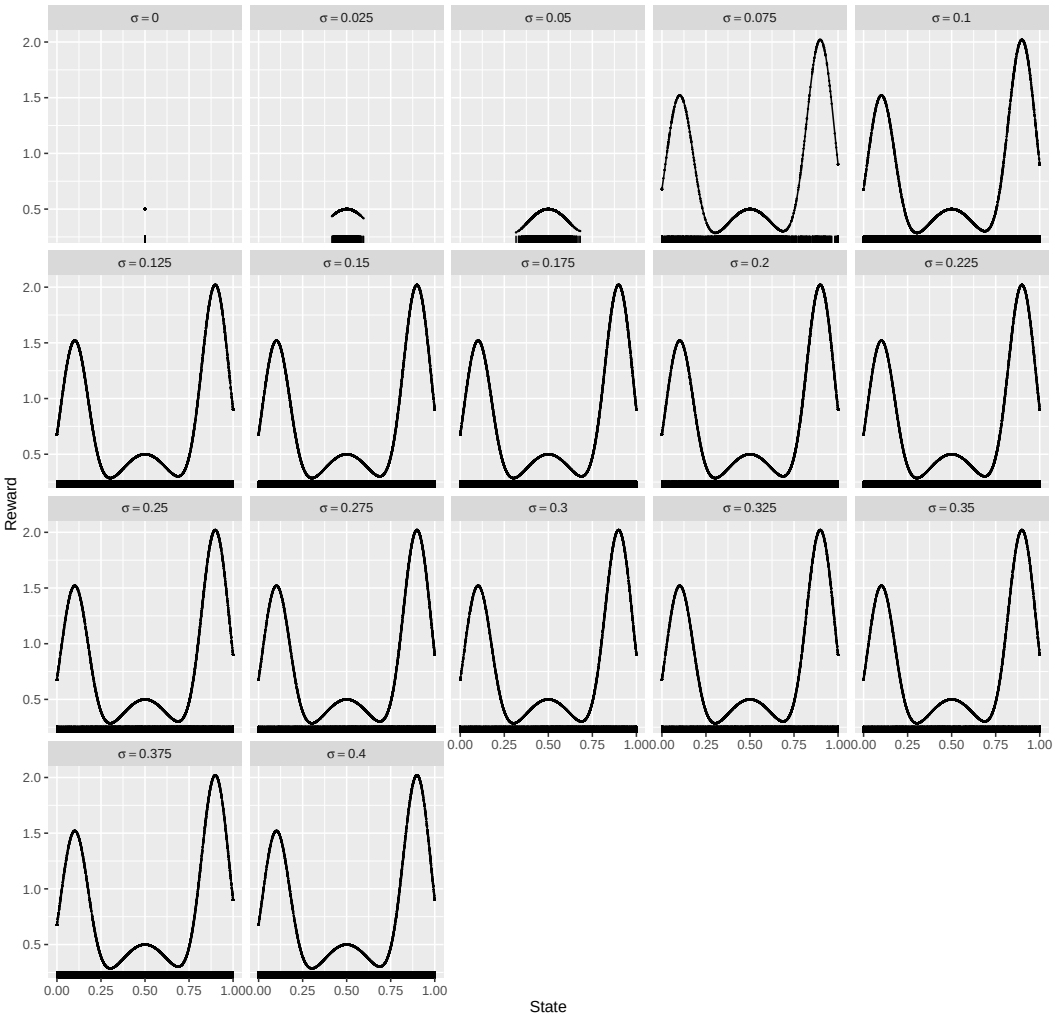

Figure 11.2: Couples (state reached, reward observed) for 50 different runs for different values of σ. For a large enough σ, the system naturally explores the space even when we play greedily i.e. try to keep the system on centered position 0.5

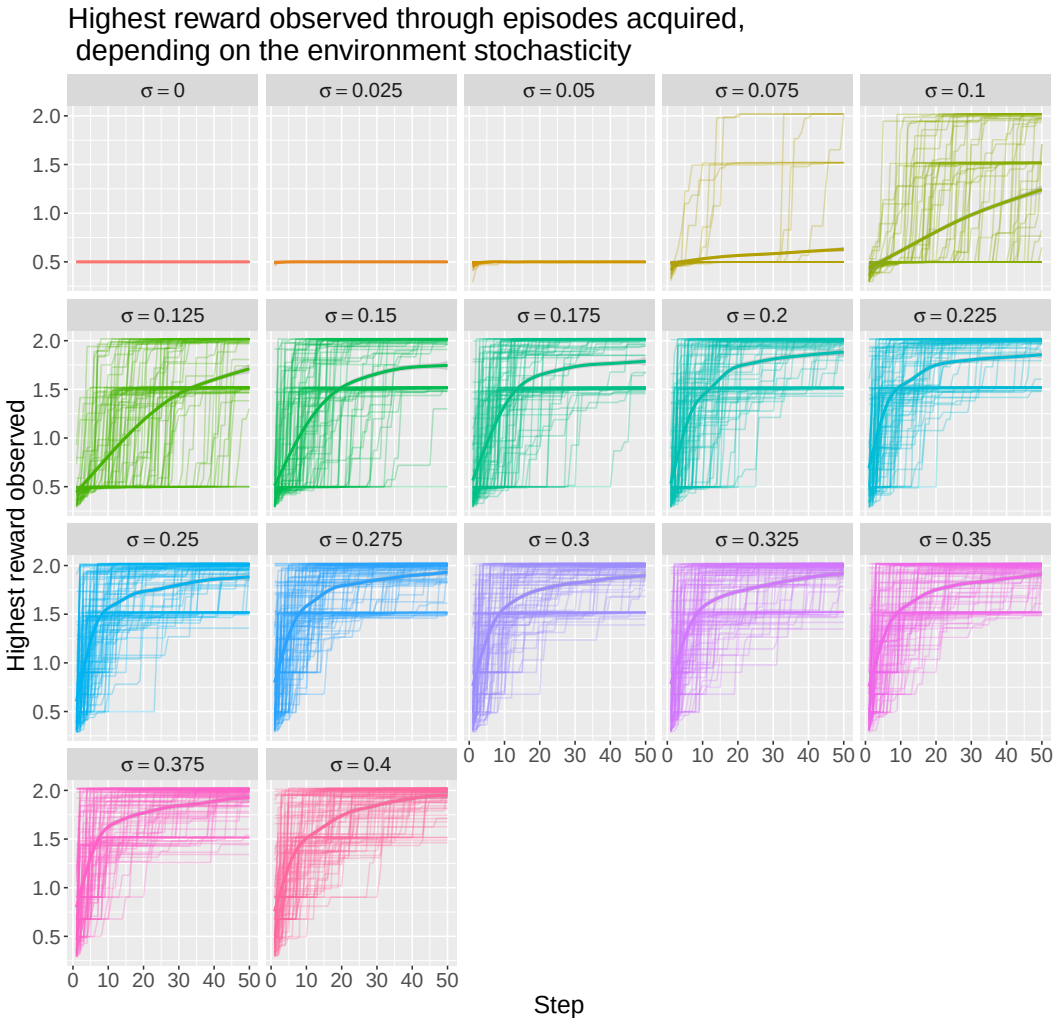

Figure 11.3: Highest reward observed throughout steps and over different runs with different parameters σ. Note that the three modes of the reward function ($\mathbf{R}(0.1) \approx 1.5, \mathbf{R}(0.5) \approx 0.5, \mathbf{R}(0.9) \approx 2$) appear quite neatly on the graphs and as σ increases, we tend to focus on higher values.

Those two policies are basic references of how much reward we could receive, one in the case the reward function were perfectly known and the other if we would focus on initial optimum point known, a secure choice.

With this last visualization we can clearly identify three phases over the values of σ:

- for σ values up to 0.05: the local maximum is the initial position, hence for low values of σ we should keep at this point, hence when σ grows, we just have some trouble keeping on that point although we are trying to ;
- for σ values from 0.05 to 0.175: we are progressively exploring (forced by the randomness of transitions still) the state space, and the cumulative reward rises quite rapidly - fast transition between 0.075 and 0.1;
- for σ values above 0.175: although the average reward diminishes only slightly with σ, the variance of the reward between different runs clearly diminishes.

11.3 Conclusion

This decision process is yet another example here intrinsic stochasticity of the system can take care, at least partly, of the exploration issue. As such, even a greedy policy, often frowned upon, performs reasonably well.

Specifically, there is a range of stochasticity level where the level is high enough that it allows for natural exploration of the state space, yet low enough that the decision we take still impacts the system.

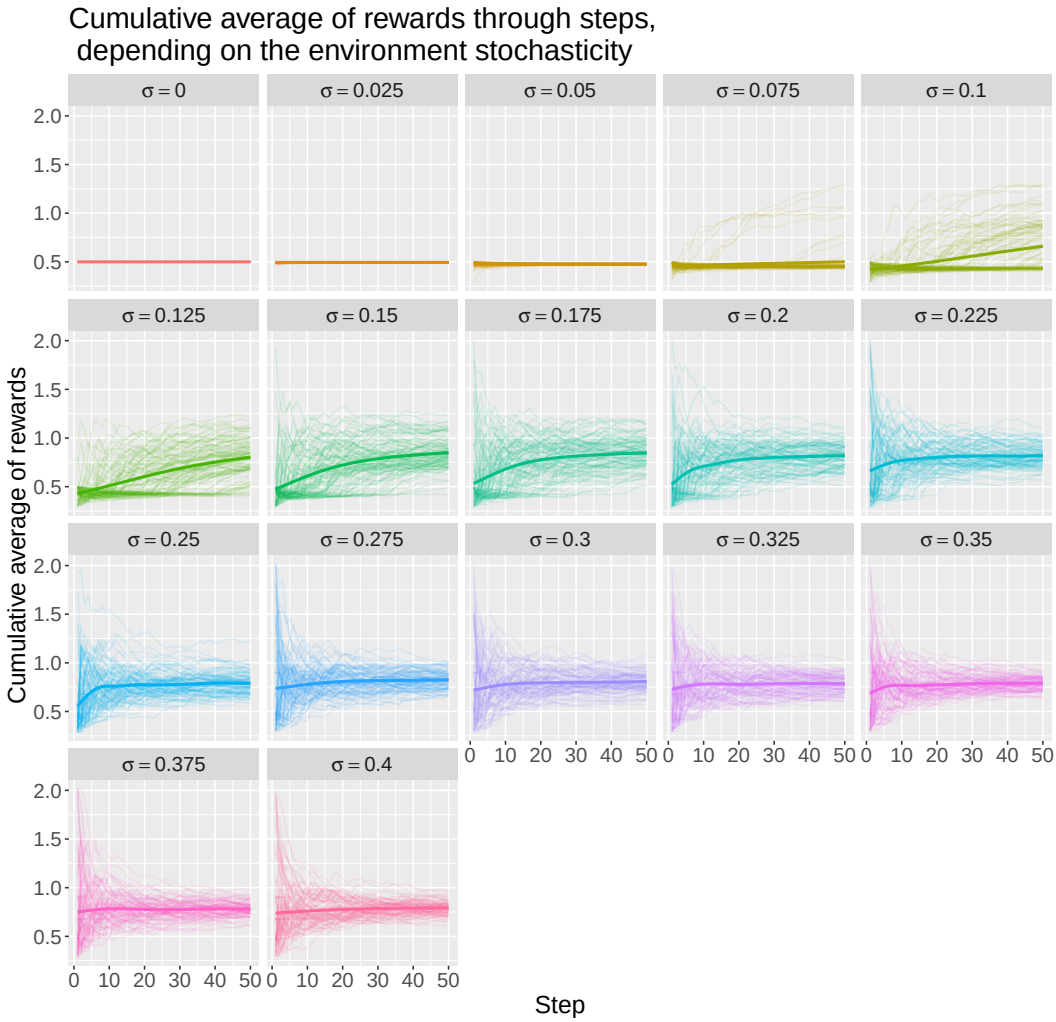

Figure 11.4: Cumulative average of rewards throughout 50 time steps for the greedy policy, depending on parameter σ. As σ increases, we can witness that we progressively reach more rewarding states overall, especially up to $\sigma = 0.175$; for higher values it appears that the variance decreases but the average effect seems constant and even potentially lowering.

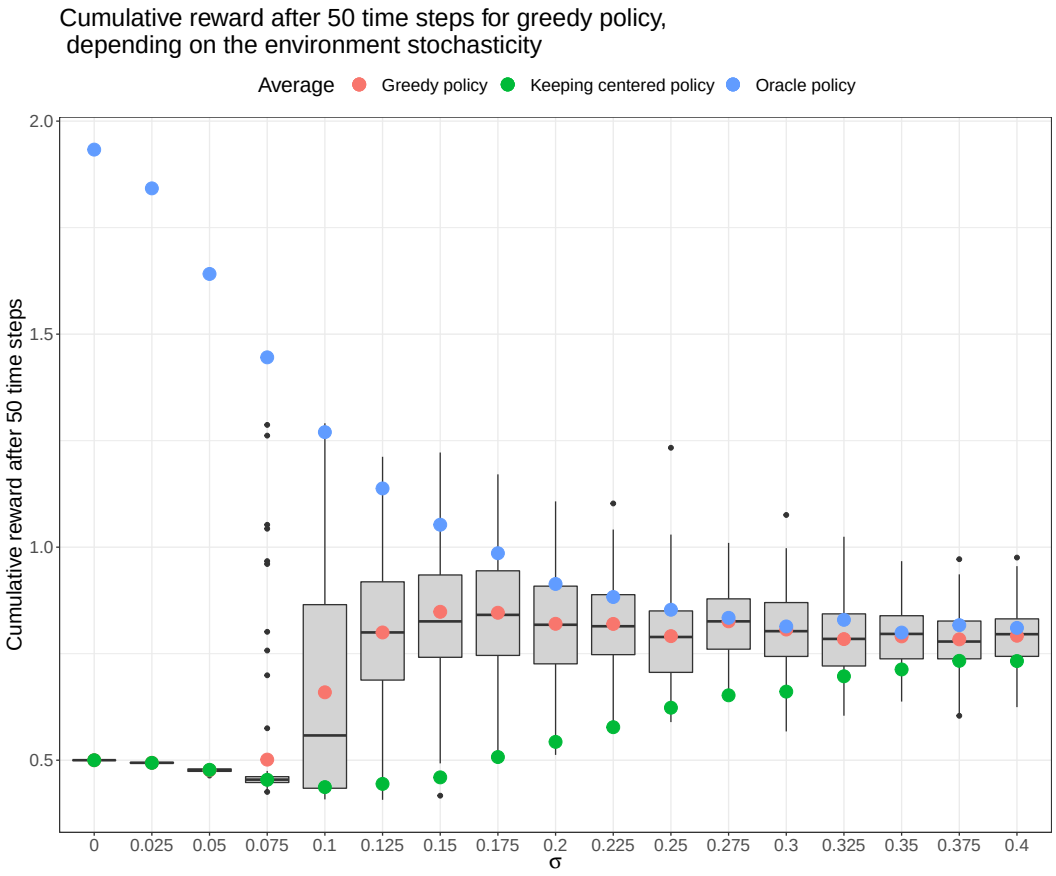

Figure 11.5: Cumulative reward after 50 time steps for the greedy policy, depending on parameter σ. One can note the clear shift in performance of the greedy strategy for σ between 0.1 and 0.25 when compared to the two baseline policies: the system is sufficiently stochastic that we are able to learn of states with higher rewards without explicit exploration **but** it is not so stochastic that the control we have over the system is negligeable, which is what is starting to happen for $\sigma = 0.4$.

Chapter 12

Contextual Bandit with linear reward function

Summary

Linear contextual bandits have been studied extensively for the past ten years. To handle such problems, current state-of-art strategies provide high-probability regret bounds of order $\tilde{\mathcal{O}}(\sqrt{Td})$, where T is the number of rounds and d the context dimension. Recent works showed that quasi-greedy strategies achieve those same bounds in specific instances of this bandit problem Bastani et al. [2017], Kannan et al. [2018]. In here, we show that those conclusions hold under a diversity assumption generalizing the one used in previous works.

About notations within this section. For any $n \in \mathbb{N}^*, [n] \overset{\Delta}{=} \{1, \dots, n\}$. Let x and y be two d-dimensional vectors and M be a $d \times d$ matrix. The scalar product between x and y will simply be written $x'y$. The L^p norm of x shall be written $||x||_p$ and $||x||_M^2 \overset{\Delta}{=} x'Mx$. For any matrix M, $\lambda_{\min}(M)$ and $\lambda_{\max}(M)$ will denote its lower and higher eigenvalues. When introducing bounding constants, we shall use the notations $\bar{\cdot}$ and $\underline{\cdot}$. Lastly, $0_{n\times p}$ will denote a zero-filled $n \times p$ matrix.

In section 12.2, after introducing the problem, the notion of regret and our assumptions, we introduce the pillar assumption to this work and take the opportunity to specify the distinctions from preceding works. Then we present our main contribution, that is high-order bounds for cumulative regret and almost sure bounds for expected regret, in the one-dimensional case, in section 12.3, and in the general case in section 12.4. In section 12.5, we present two key results on our model's estimator, which are crucial to the regret proofs. Technical details of the proofs are reported in section 12.6 and follows the conclusion in section 12.7.

12.1 Introduction

Consider the following game: an agent is presented with $K \geq 2$ different options, picks one and observes some feedback. This experiment is repeated for some time T, potentially random. In linear contextual bandits, the options - *arms* - are characterized by a vector, the *context*. It is assumed that the *rewards* - the feedback the agent receives - are linear in expectation with respect to those context vectors.

Several strategies have been proposed to deal with those bandits, such as SUPLINREL Auer et al. [2002], SUPLINUCB Chu et al. [2011] and OFUL Abbasi-Yadkori et al. [2011]. Those strategies rely on the Optimism Under the Face of Uncertainty principle, which dictates playing arms with high upper-confidence bounds i.e. where potentially there would be most reward. The Bayesian approach has also been applied, see Agrawal and Goyal [2013]. Both streams of literature focus on efficient exploration-exploitation balancing, which is at the core of online learning. This need for a trade-off is easily demonstrated with the classic K-Arm Bandit problem, for which greedy strategies lead with non-zero probability to linear regret.

Yet, several papers showed that a greedy policy can be competitive with state-of-art methods in terms of regret bounds [see Bastani et al., 2017, Kannan et al., 2018]. In table 12.1 we reported the order of high-probability regret bounds for UCB, Thompson Sampling and a Greedy approach. Those orders are competitive.

However, in both papers [Bastani et al., 2017, Kannan et al., 2018] the authors added assumptions on the problem itself, which limit practical applications of such models. In this paper we try to generalize their results to the generic linear contextual bandit problem. In particular, we prove that a Greedy-like strategy achieves a high-probability bound of order $\tilde{\mathcal{O}}(\sqrt{dT})$.

APPROACH	BOUND ORDER
UCB Chu et al. [2011]	$\tilde{\mathcal{O}}(\sqrt{Td})$
TS Agrawal and Goyal [2013]	$\tilde{\mathcal{O}}(d^2/\varepsilon\sqrt{T^{1+\varepsilon}})$
Greedy Kannan et al. [2018]	$\tilde{\mathcal{O}}(\sqrt{Td})$

Table 12.1: Some high-probability bounds for cumulative pseudo-regret from the literature of stochastic linear contextual bandits.

12.2 Context

Let us present more formally the bandit problem, the key assumption we make, define our performance metrics, and finally our algorithm.

12.2.1 Linear contextual bandit

Let us give a description of the linear contextual bandit, following Chu et al. [2011].

Consider the following game. At time $t \in [T]$, the agent observes for each arm $a \in [K]$, some context $X_{ta} \in \mathbb{R}^d$, $d \geq 1$. The context set $\mathbf{X}_t \overset{\Delta}{=} \{X_{ta}; a \in [K]\}$ is generated from distribution $\mathcal{P}$ with support $\mathcal{X}$, a $K \times d$ space. Based on this contextual information, the agent chooses arm $A_t \in [K]$ and collects reward R_{tA_t} on which we impose the following linearity assumption.

Assumption 1. $\exists! \beta \in \mathbb{R}^d \setminus \{0_d\} : \forall t \in [T], a \in [K],$

$$\mathbb{E}(R_{ta}|X_{ta}) = X'_{ta}\beta. \tag{12.2.1}$$

The goal of this game is to maximize the cumulative sum of expected rewards

$$\sum_{t=1}^{T} X'_{tA_t}\beta. \tag{12.2.2}$$

Remark 1. *The K-Arm Bandit problem is a specific instance of this problem, taking β to be the vector of expected rewards of the arms and setting $\mathbf{X}_t = \mathbf{diag}(K)$, which allows no information transfer.*

We write $X_t \triangleq X_{tA_t}$ and $R_t \triangleq R_{tA_t}$. Let the σ-algebra

$$\mathcal{F}_{t+1} \triangleq \sigma\{\mathbf{X}_1, A_1, R_1, \mathbf{X}_2, \ldots, R_t, \mathbf{X}_{t+1}\} \tag{12.2.3}$$

represent the knowledge of the agent right before his decision at time $t+1$.

Let us now add important assumptions for the rest of our paper.

Assumption 2. $\exists \sigma \in (0; +\infty) : \forall t \in [T], a \in [K], \varepsilon_{ta} \triangleq R_{ta} - \mathbb{E}(R_{ta}|X_{ta})$ *is σ sub-Gaussian conditionally to* $\{\mathcal{F}_t, (\varepsilon_{k\tilde{a}}; k \in [t-1], \tilde{a} \in [K])\}$.

Definition 1. *A random variable X is σ sub-Gaussian if, for any $\gamma > 0$, $\mathbb{E}(\exp(\gamma X)) \leq \exp(\sigma^2\gamma^2/2)$.*

Assumption 3. $\exists \bar{\lambda} \in (0; +\infty)$:

$$\sup_{\mathbf{X}_1} \sup_{a \in [K]} ||X_{1a}||_2^2 < \bar{\lambda}. \tag{12.2.4}$$

Remark 2. *Please note that the dimension factor d is hidden within $\bar{\lambda}$.*

Let us make a few remarks on those assumptions, which are quite common in the contextual bandit literature. The first one specifies how information is shared, the last two help us derive statistical properties of our model estimator, which is introduced in section 2.4.

12.2.2 Bound on lower eigenvalue

Since the contexts are random variables and the final selected context X_t depends on previous contexts via the policy followed, the sequence $\{X_t; t \in [T]\}$ is not *independent and identically distributed* (iid).

Let us recall an important result regarding the convergence of the OLS estimate in a regression model where the iid assumption is not fulfilled by the data.

Lemma 1 (Lai and Wei, 1982, Corollary 3)**.** *Consider the regression model $R_k = X'_k\beta + \varepsilon_k$, $k \in [t]$ where $\{\varepsilon_k; k \in [t]\}$ is a martingale difference sequence with respect to an increasing sequence of σ-fields $\{\mathcal{F}_k; k \in [t]\}$ such that $\sup_{k\in[t]} \mathbb{E}(\varepsilon_k^2|\mathcal{F}_k) < +\infty$ almost surely and X_k is $\mathcal{F}_k$-measurable. Writing $M_t = \sum_{k\leq t} X'_k X_k$ for any integer t, assume that*

$$\lim_{t\to+\infty} \lambda_{\min}(M_t) = +\infty \text{ and}$$
$$\lim_{t\to+\infty} \frac{(\log \lambda_{\max}(M_t))^{1+\alpha}}{\lambda_{\min}(M_t)} = 0 \text{ for some } \alpha > 0.$$

Then $\hat{\beta}_t$ converges almost surely to β.

Based on Assumption 2 the requirement on the error terms of the model is checked. To fulfill the last two requirements, a sufficient condition would be to have $\lambda_{\min}(M_t)$ of order $(\log(t))^{1+\alpha}$ for some $\alpha > 0$. Indeed, from Assumption 3, $\lambda_{\max}(M_t)$ is upper bounded by $t\bar{\lambda}$.

Consider, for a specific policy, the following scalar:

$$\underline{\lambda} \triangleq \inf_{\hat{\beta} \in \mathbb{R}^d} \lambda_{\min} \left(\mathbb{E}\left[X_1 X_1' \middle| \hat{\beta} \right] \right) \tag{12.2.5}$$

where X_1 is the context vector selected following the policy, with our estimate $\hat{\beta}$.

Assumption 4. *Under the context distribution $\mathcal{P}$, $\underline{\lambda} > 0$.*

This assumption ensures that $\lambda_{\min}(M_t)$ is of order t with high-probability, based on concentration results from random matrix theory, which we will detail in the following section. Those results ensure the convergence of the OLS estimate but also control the order of the regret.

Assumption 4 and the Multi-Armed Bandit problem In the Multi-Armed Bandit Bandit problem, there is no information sharing nor context diversity ; specifically, $\mathbf{X}_t = \mathbf{diag}(K)$, β is a vector of length K, where the a-th element is the expected reward of arm a, for any $a \in [K]$.

Now, whatever the arm chosen A_t, we have

$$X_t = \overset{\qquad\qquad A_t}{\begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}} \tag{12.2.6}$$

and therefore $X_t' X_t$ is a $K \times K$ matrix filled with zeros everywhere but at coordinate (A_t, A_t):

$$X_t' X_t = \begin{array}{c} \\ A_t \end{array} \overset{A_t}{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}} \tag{12.2.7}$$

Whatever (K, a), the lowest eigenvalue of matrix $X_t' X_t$ (which is deterministic in the Multi-Armed Bandit problem) is zero i.e. $\underline{\lambda} = 0$.

This transcribes quantitatively the fact that when we draw an observation from an arm within the Multi-Armed Bandit problem, we only learn on the reward distribution associated to this particular arm. We will focus on linear contextual bandits where information sharing does occur, hence the previous assumption.

Assumption 4 in the literature In the works of Bastani et al. [2017] and Kannan et al. [2018], Assumption 4 is justified in two very different manners. In Bastani et al. [2017], the authors assume that the contextual information is the same for all arms, but the coefficients of β are indexed over the arms:

$$\forall t \in [T], a \in [K], \; R_{ta} = X_t' \beta_a + \varepsilon_{ta} \tag{12.2.8}$$

which is a special instance of the model we study. They justify Assumption 4 by stating that with enough context variance, whatever the set of estimates $(\hat{\beta}_a; a \in [K])$ all are bound to be played some time. In Kannan et al. [2018], the authors assume that the contexts are arm-dependent and

chosen by an adversary up-to some independent random perturbations. The level of perturbation introduced increases the value of $\underline{\lambda}$ and ensures that we learn properly.

Here, we consider the generalized setting without imposing a particular structure on the context distribution and simply take the assumption that there is a positive lower bound on this quantity, which is realistic as preceding works have illustrated.

12.2.3 Performance metrics

As the goal of the game is to maximize expected cumulative rewards, we may define the optimal arm at time t by

$$A_t^* \triangleq \arg\max_{a\in[K]} X_{ta}'\beta. \tag{12.2.9}$$

To evaluate strategies, the bandit community uses regret as a standard metric. Regret measures the discrepancy between the expected rewards obtained when picking arms $\{A_t; t \in [T]\}$ rather than the oracle $\{A_t^*; t \in [T]\}$.

Definition 2. *The cumulative pseudo regret suffered up to time T is*

$$\mathbf{regret}(\mathcal{F}_T) \triangleq \sum_{t=1}^{T} \max_{a\in[K]} X_{ta}'\beta - X_{tA_t}'\beta. \tag{12.2.10}$$

Definition 3. *The cumulative expected pseudo regret suffered up to time T is*

$$\mathbb{E}_{(\mathbf{X}_1,\dots,\mathbf{X}_T)}\left[\mathbf{regret}(\mathcal{F}_T)\right]. \tag{12.2.11}$$

We note the supremum over the instantaneous regret

$$\bar{r} \triangleq \sup_{\mathbf{X}_1} \sup_{(a,\tilde{a})\in[K]^2} X_{1a}'\beta - X_{1\tilde{a}}'\beta. \tag{12.2.12}$$

From Cauchy-Schwarz, $\bar{r} < 2||\beta||_2\bar{\lambda}^{1/2}$.

In the main text we will directly write regret or expected regret and not mention *cumulative/pseudo* prefixes.

In some cases, it will be interesting to directly look at the decision made, and check its optimality:

$$\mathbf{correctAction}(\mathcal{F}_T) \triangleq \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}\{A_t = A_t^*\}. \tag{12.2.13}$$

12.2.4 ExploitASAP: a greedy algorithm

Given the definition of optimal arms (equation 12.2.9), a natural approach is to take

$$A_t = \arg\, max_{a\in[K]}\ X_{ta}'\hat{\beta}_t \tag{12.2.14}$$

where $\hat{\beta}_t$ is an estimate of β based on past data. Given Assumption 1, a natural choice for the estimator is the Ordinary Least Squares (OLS) estimator defined as

$$\hat{\beta}_t = M_t^{-1}\big(\sum_{k=1} t - 1X_kR_k\big) \tag{12.2.15}$$

where

$$M_t \triangleq \sum_{k=1} t - 1 X_k X_k'. \tag{12.2.16}$$

Algorithm 14, EXPLOITASAP follows this greedy decision-making process. Now, there are two situations which need to be handled.

First, ties might occur when evaluating our criterion, especially if $\mathcal{X}$, the support of $\mathbf{X}_t$ is discrete. We propose to draw uniformly at random from the pool of maximizers.

Second, the OLS estimator is defined only when M_t is invertible. As such, in the first time steps of the game, we propose to rely on some prior $\beta_0 \in \mathbb{R}^d$ which acts as temporary substitute. If such prior is unavailable, then one can set $\beta_0 = 0_{d\times 1}$, in which case we will pick uniformly each arm.

Note that when following this algorithm, the agent needs not know $\bar{\lambda}$, σ or T: there is no parameter tuning involved.

Unsurprisingly, this algorithm has similarities with many greedy algorithms introduced in the literature, such as LINGREEDY in Chou et al. [2015] which relies on a Ridge estimator. Interestingly, the authors provided simulations results which concur that Greedy strategies are far from sub-optimal compared to SUPLINUCB or their approach, LINPRUCB.

Algorithm 14: ExploitASAP algorithm

Initialize: $M_1 \leftarrow 0_{d,d}$, $U_1 \leftarrow 0_{d,1}$, exploit $\leftarrow$ **false**
Choose: $\beta_0 \in \mathbb{R}^d$
for $t = 1, \dots, T$ **do**
  /* Observe set of contexts */
  collect $\mathbf{X}_t$
  /* Choose arm A_t, if possible with running estimate of β */
  if exploit == **true** **then**
    $\hat{\beta}_t \leftarrow M_t^{-1} U_t$
    $A_t \leftarrow \arg\max_{a\in\{1,\dots,K\}} X_{ta}' \hat{\beta}_t$ (†)
  else
    $A_t \leftarrow \arg\max_{a\in\{1,\dots,K\}} X_{ta}' \beta_0$ (†)
  end
  /* Collect reward */
  $R_t \leftarrow X_{tA_t}' \beta + \varepsilon_t$
  /* Update estimates */
  $M_{t+1} \leftarrow M_t + X_{tA_t} X_{tA_t}'$
  $U_{t+1} \leftarrow U_t + X_{tA_t} R_t$
  /* Check invertibility of M_t */
  if exploit == **false** and $\lambda_{\min}(M_t) > 0$ **then**
    exploit $\leftarrow$ **true**
  end
end
(†) in case of ties, pick uniformly amongst maximizers.
return $(\mathbf{X}_t, A_t, X_{tA_t}, R_t)_{t=1}^T$

12.3 The one-dimensional case

12.3.1 Intuition

Let us consider the one-dimensional case of the bandit problem introduced above. The objective, at any time t, is to pick the arm which maximizes the expected return i.e. find

$$A_t^* \triangleq \arg \max_{a \in \{1,\dots,K\}} \beta X_{ta} \tag{12.3.1}$$

which is equivalent to estimating the sign of unknown parameter β.

Writing $\hat{\beta}_{t-1}$ an estimate of β based on acquired data up to time t, we are interested in minimal assumptions such that playing greedily based on estimate $\hat{\beta}_{t-1}$ is still guaranteed to have proper regret bounds.

Let A_t denote the arm selected at time t, we consider the following quantity, which is a key indicator of the diversity of the stochastic contexts:

$$\underline{\lambda} \triangleq \inf_{\hat{\beta} \in \mathbb{R}} \mathbb{E}(X_{tA_t}^2 | \hat{\beta}) \tag{12.3.2}$$

$$= \min \Big\{ \underbrace{\mathbb{E}[(\min_a X_{ta})^2]}_{\text{if } \hat{\beta}<0}, \underbrace{\mathbb{E}[(\max_a X_{ta})^2]}_{\text{if } \hat{\beta}>0}, \underbrace{\sum_{a=1}^{K} \mathbb{E}(X_{ta}^2)\pi(a)}_{\text{if } \hat{\beta}=0} \Big\} \tag{12.3.3}$$

where π is the baseline set of probabilities of drawing the arms. A typical baseline is the uniform distribution.

The quantity $\underline{\lambda}$ basically indicates that whatever the current estimate, the context observed playing greedily will have minimal variance, and as long as $\underline{\lambda}$ is strictly positive, we will gain information and improve on our estimate of β and therefore be able to make better decisions. Before going on to proofs, let us consider the following case: imagine one of the arms has a fixed value (highest negative or highest positive point). Then, if we are unlucky in our first estimates (because of estimation error), we might get stuck into picking this arm (which is in fact the worst one to pick) for all succeeding steps and we won't know any better since we essentially don't learn.

Let us illustrate this with two simple models:

Model 1: four arms, with independent distributions of context: for any $t \in \mathbb{N}$,

$$\forall a \in [4], \quad X_{ta} \sim \text{Gaussian}(\mu_a, \nu) \tag{12.3.4}$$

with $\mu = (-4, 0, 3, 4)$ and ν was tested for values in $\{0, 1, 5\}$. An example of sample distributions for the context is given in figure 12.1. We take $\beta = 5$, $\sigma = 500$.

Model 2: two arms, with independent distributions of context: for any $t \in \mathbb{N}$,

$$\forall a \in [2], \quad X_{ta} \sim \text{Gaussian}(\mu_a, \nu) \tag{12.3.5}$$

with $\mu = (0, 1)$ and ν was tested for values in $\{0, 1, 5\}$. We took $\beta = 5$, $\sigma = 50$.

In the first model, following a greedy decision-making, we will almost surely pick the highest or lowest context, depending on the sign of our estimate. Since contexts are non-zero almost surely, the quantity $\sum_t X_t^2$ grows and one can show it ensures convergence of the estimate. In the second model however, we can fall in the trap where we pick the first arm (because we have been led to

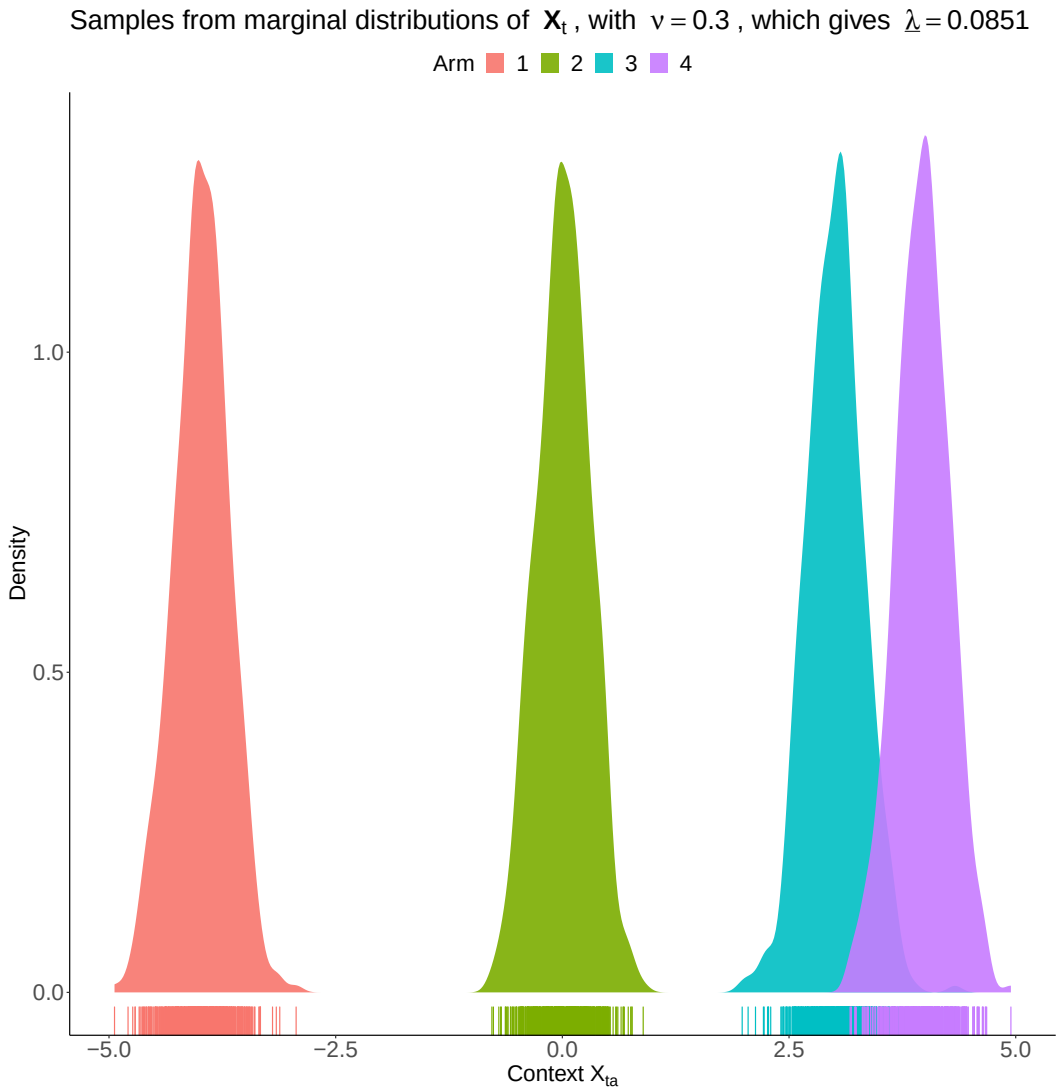

Figure 12.1: Samples from context marginal distributions for $\nu = 0.3$. Note that for this variance level, $\underline{\lambda} \approx 0.08$.

believe that β is negative although it is not) and we actually cannot update our estimate of β from then on, since:

$$\hat{\beta}_t = \frac{\sum_{j \leq t} X_j Y_j}{\sum_{j \leq t} X_j^2} \tag{12.3.6}$$

hence all observations where X_j is 0 are not used.

We experimented with both models, applying the greedy strategy as well as a `Randomized Draws` strategy: assuming we know the noise terms are Gaussian, one may show that

$$\hat{\beta} = \beta + \frac{\sigma}{\sum_j X_j^2} \text{Gaussian}(0, 1) \tag{12.3.7}$$

in this one-dimensional case. As such, one could pick

$$A_t = \arg\max_a X_{ta} \beta_{\text{sample}}, \tag{12.3.8}$$

where

$$\beta_{\text{sample}} \sim \hat{\beta} + \frac{\sigma}{\sum_j X_j^2} \text{Gaussian}(0, 1). \tag{12.3.9}$$

This would allow exploration part of the time, similarly to the e-greedy strategy, but at least it is guided by our uncertainty levels.

Note that it is (almost) equivalent to picking the maxima of contexts with probability

$$\mathbb{P}\left(\hat{\beta} + \frac{\sigma}{\sum_j X_j^2} \text{Gaussian}(0, 1) > 0\right) \tag{12.3.10}$$

and the minima of contexts otherwise.

For the first model we simulated 100 sequences of total length 5000, during which we focused on the proportion of sequences where the proper sign was estimated (figure 12.2) and the cumulative regret (figure 12.3).

It appears that drawing samples at random helps control the observed regret, as less sequences tend to get high cumulative regret values following this strategy, compared to the greedy one. However, performances are on par in average, whatever the variance level of the context values.

For the second model we simulated 50 sequences of total length 500, during which we focused on the proportion of sequences where the proper sign was estimated (figure 12.4) and the cumulative regret (figure 12.5).

As we have seen before, even with non-noisy observations, we are still learning. However, if we systematically pick the context in 0, ($\nu = 0$ setting), we observe data uninfluenced by β and only pure noise ; as such our estimate of β cannot converge to its true value, which is why three out of four of the sequences did not manage to retrieve the proper sign of β **and** get stuck on their estimate. For those, a linear regret appears quite neatly on the cumulative regret graphs.

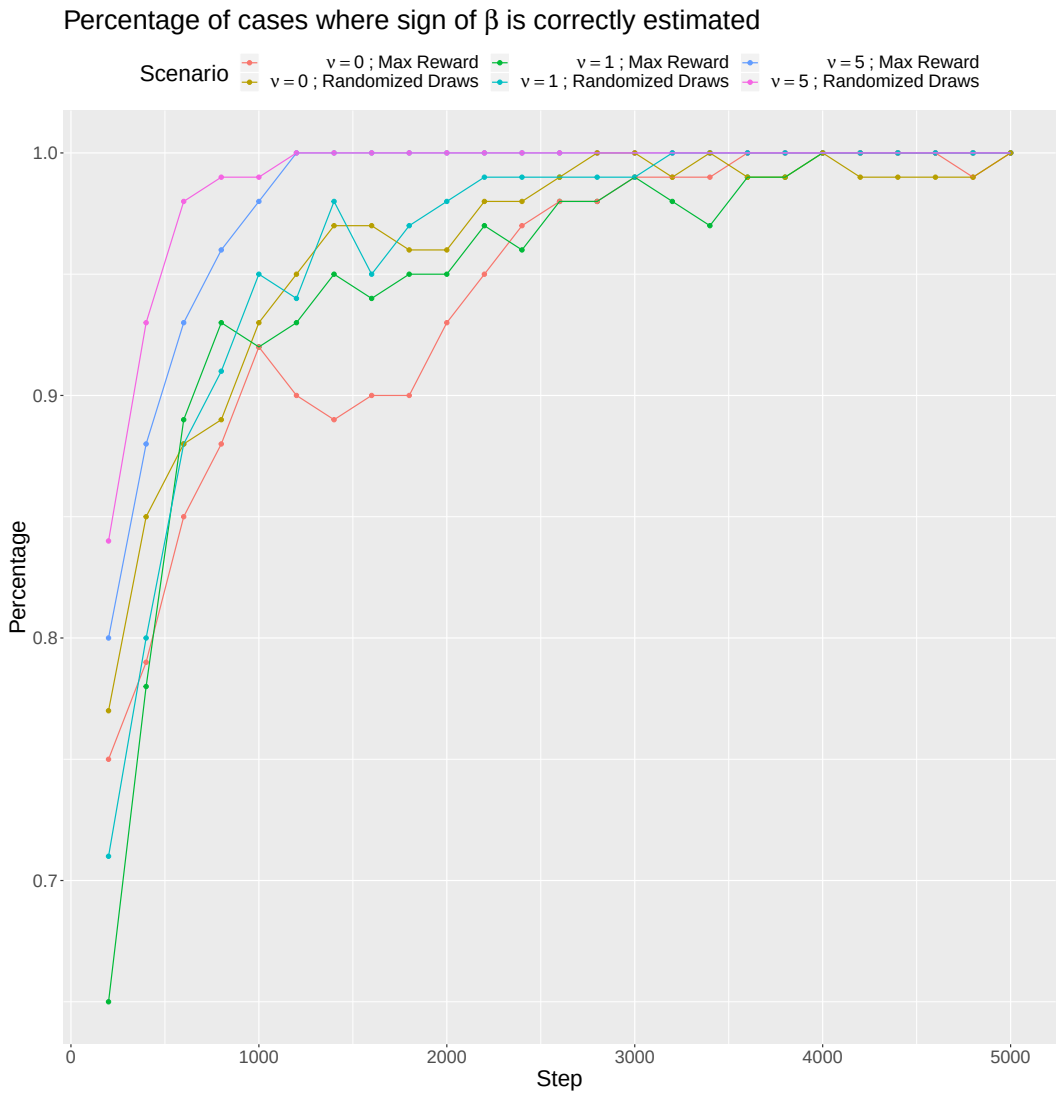

Figure 12.2: Percentage of cases over 50 simulations where the correct sign was found throughout 500 steps for first model.

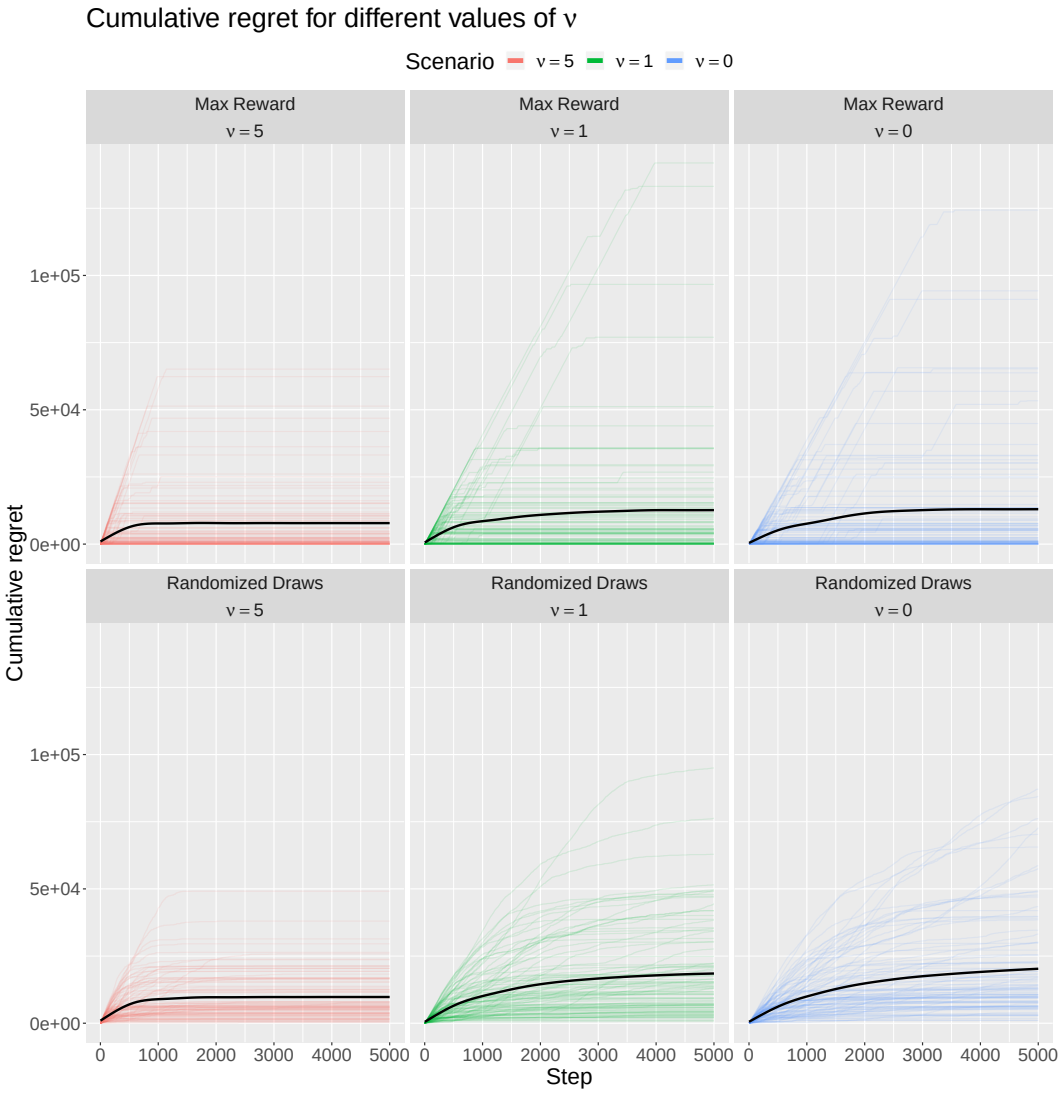

Figure 12.3: Cumulative regret observed for first model on 50 different simulations throughout 500 steps. Average regret is shown as the bold line for each scenario of ν and strategy.

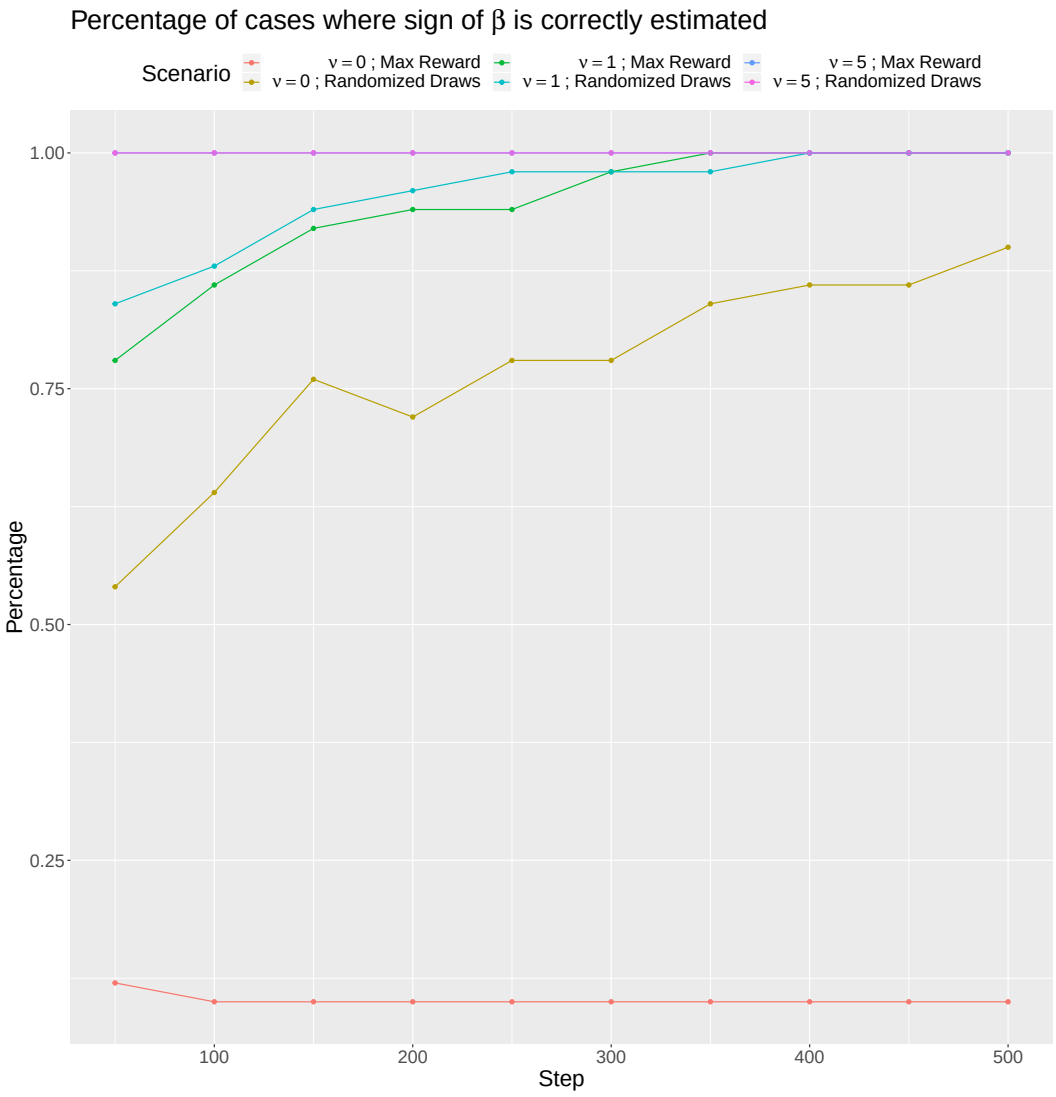

Figure 12.4: Percentage of cases over 50 simulations where the correct sign was found throughout 500 steps for second model.

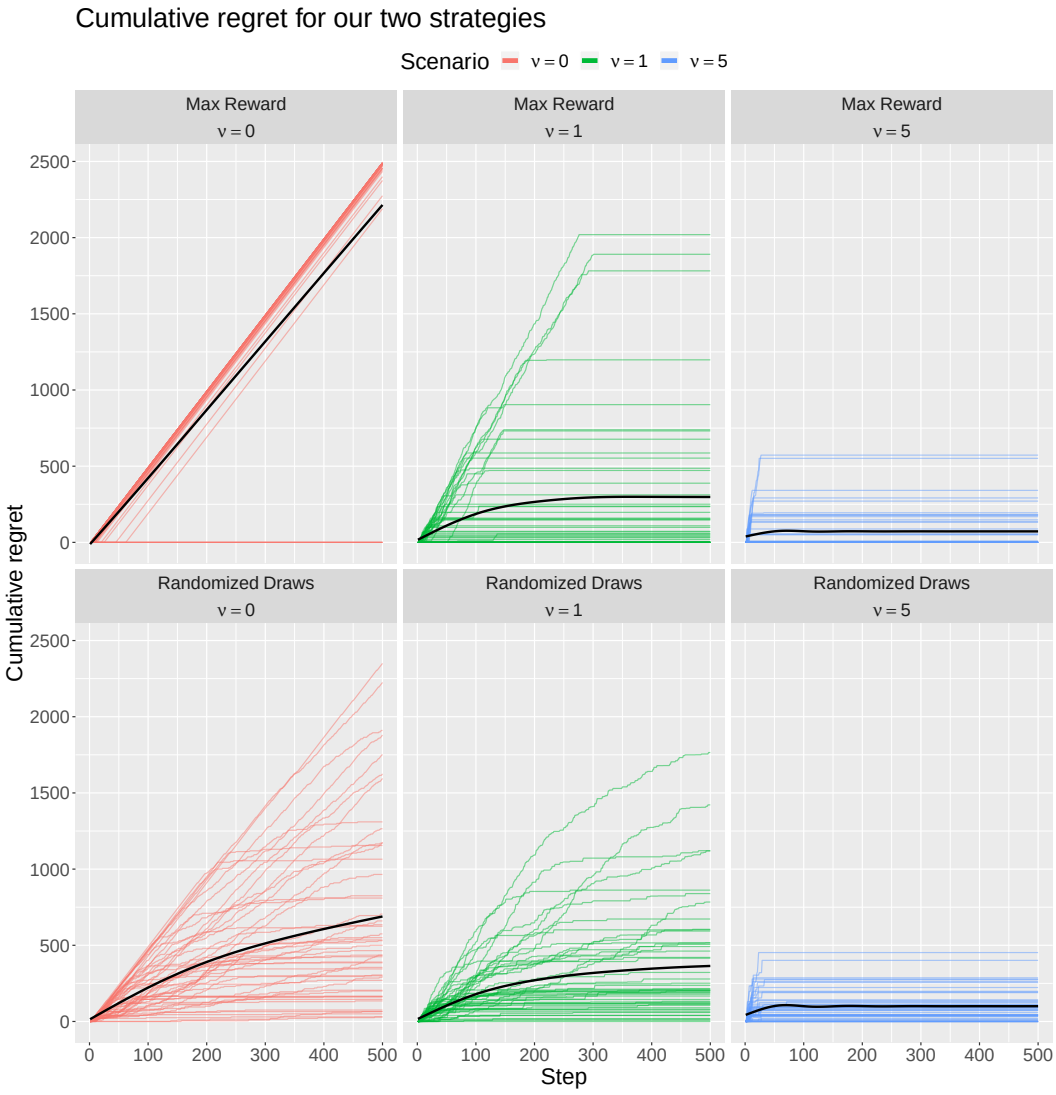

Figure 12.5: Cumulative regret observed for second model on 50 different simulations throughout 500 steps. Average regret is shown as the bold line for each scenario of ν and strategy.

12.3.2 Regret

We define, for all $\gamma \in (0,1)$,

$$h(\gamma) \triangleq \frac{\exp(-\gamma)}{(1-\gamma)^{1-\gamma}}, \tag{12.3.11}$$

for all $\gamma \in (0,1)$,

$$\kappa(\gamma) = \sqrt{2 + \log\left(1 + \frac{2\bar{\lambda}}{\underline{\lambda}(1-\gamma)}\right)} \tag{12.3.12}$$

and for all $\delta_2 \in (0,1), \gamma \in (0,1)$,

$$t_0(\delta_2, \gamma) \triangleq \max\left\{d; \frac{\bar{\lambda}\log(d\log(T/d)/(\log(2)\delta_2))}{\underline{\lambda}\log(1/h(\gamma))}\right\}. \tag{12.3.13}$$

which will appear in several results.

Theorem 1. *We take assumptions 1-4. Following algorithm 14 in the one-dimensional case, the cumulative regret is at most*

$$\bar{r}\max\left\{t_0(\delta_2,\gamma); \frac{\sigma^2}{\beta^2}\frac{\bar{\lambda}}{\underline{\lambda}}\frac{4\kappa(\gamma)^2\log(1/\delta_1)}{(1-\gamma)}\right\} \tag{12.3.14}$$

with probability $1-(\delta_1+\delta_2)$.

This bound is at least of the order $\log(\log(T))$ and if the ratio σ^2/β^2 is too large we might pay more.

Proof sketch. The proof of this result essentially comes from two ingredients: first, realizing that selecting the optimal arm in the one-dimensional case is equivalent to knowing the sign of β. Second, we apply jointly lemmas 2 and 4 to get the desired result. The complete proof is in the supplemental material. □

Let us define, for $d = 1$,

$$\tilde{r} \triangleq |\beta|\mathbb{E}_{\mathbf{X}_t}\left[\max_a X_{ta} - \min_a X_{ta}\right]. \tag{12.3.15}$$

Evidently, $\tilde{r}$ is much lower than $\bar{r}$.

Theorem 2. *We take assumptions 1-4. Following algorithm 14 in the one-dimensional case, the cumulative expected regret is at most*

$$\tilde{r}\left(2 + \max\left\{t_0\left(1/T,\gamma\right); \frac{\sigma^2}{\beta^2}\frac{\bar{\lambda}}{\underline{\lambda}}\frac{2\kappa(\gamma)\log(T)}{(1-\gamma)}\right\}\right).$$

The proof relies on very similar arguments than that of theorem 1 and is given in the appendix. Interestingly, when taking the expectation over all possible contexts, we get a higher bound than in the worst case setting: it is essentially scaled up by a $\log(T)$ factor.

12.4 The multi-dimensional case

12.4.1 Intuition

For anyone interested, we openly provide an R Shiny application which allows users to run linear contextual bandits and compare regret and expected regret for ExploitASAP and LinUCB. The

users can choose bandit parameters (K, T, σ, d, β). They can also pick context generation amongst two cases: *K-Arm Bandit* or *Independent Truncated Gaussians*. In this second scenario, for all $a \in [K]$, $t \in [T]$, X_{ta} follows distribution $\max\{\min\{\text{Gaussian}(0,1); +4\}; -4\}$. One can also pick LINUCB exploration coefficient α and the simulation parameters: the *seed*, for reproducibility and the number of simulations N.

To extend on the previous section on one-dimensional setting we considered two scenarios: $\beta = (5, 1, 0)$ and $\beta = (3, 2, 1)$. We took the following parameters: $K = d = 3$, $T = 200$, $\sigma = 10$. In figure 12.6, we plotted the cumulative regret for K-arm bandit problem with the two scenarios on left and right graphs. In figure 12.7, we plotted the cumulative regret for the contextual bandit problem assuming independent gaussian distributions for context, with $\beta = (5, 1, 0)$ (left) and $\beta = (3, 2, 1)$ (right).

One can clearly observe that when contexts are stochastic, the regret has a very different behavior than when contexts are deterministic, especially so when we picked $\beta = (3, 2, 1)$, which leads to linear regret in some instances because estimates are misleading and do not bring enough information during the considered time frame. As a standard of comparison, we looked at UCB strategy, which generally behaves quite well even under the K-arm bandit setting. We see that this baseline strategy and the simple greedy strategy are undiscernable in terms of regret when context is inherently stochastic.

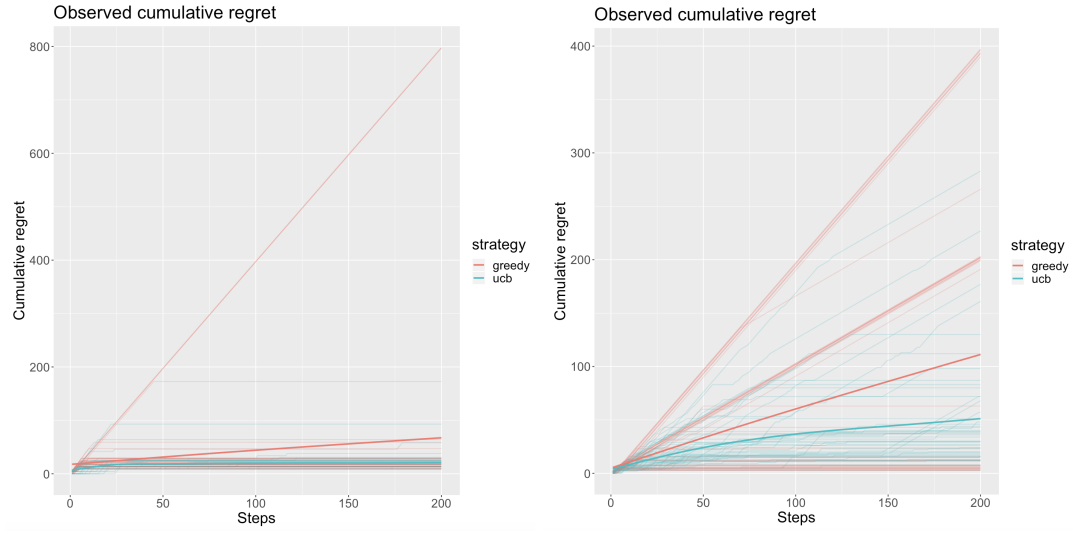

Figure 12.6: Cumulative regret for K-arm bandit problem with $\beta = (5, 1, 0)$ (left) and $\beta = (3, 2, 1)$ (right). Note that with the second instanciation of parameter vector β, in the right graph, the cumulative regret is more often linear when following the greedy strategy, and gets higher values under the UCB strategy than it did with the first instanciation of β. This is due to the fact that in the second instanciation, it is less easy to distinguish which arm is the best than in the first one.

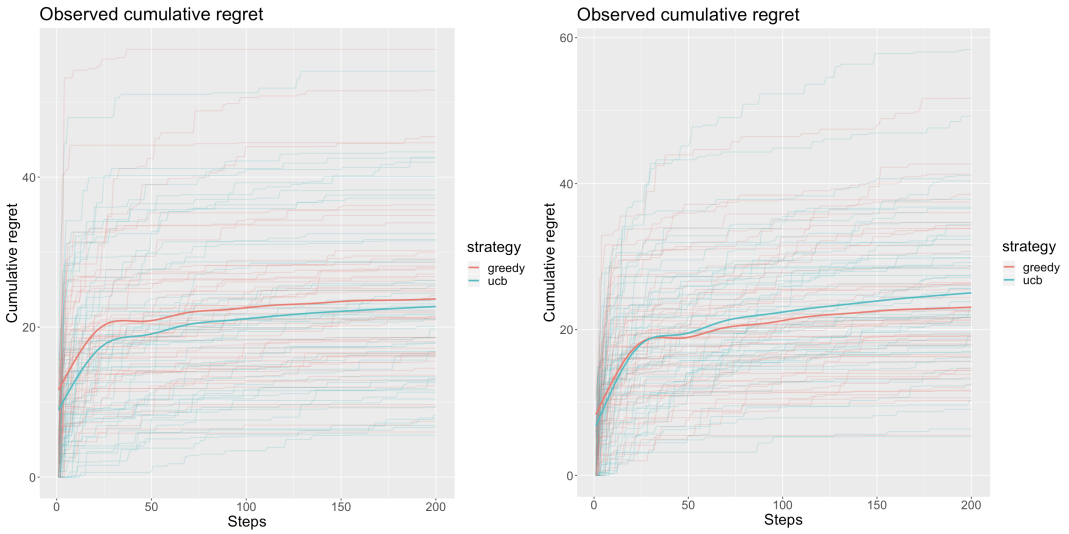

Figure 12.7: Cumulative regret for the contextual bandit problem assuming independent gaussian distributions for context, with $\beta = (5, 1, 0)$ (left) and $\beta = (3, 2, 1)$ (right). Note how cumulative regret, whatever the instanciation and the policy, follows a log-like pattern, much different from what we observed in the first figure.

12.4.2 Regret

Theorem 3. *We take assumptions 1-4. Following algorithm 14, the cumulative regret is at most*

$$\bar{r}t_0(\delta_2, \gamma) + 4\kappa(\gamma)\sigma\sqrt{\frac{\bar{\lambda}T\log(T/\delta_1)}{(1-\gamma)\underline{\lambda}}}$$

with probability $1 - (\delta_1 + \delta_2)$.

Proof. First, let's show that following a greedy strategy (i.e. $A_t \triangleq \arg\max_{a\in[K]} X'_{ta}\hat{\beta}_t$) implies that the instantaneous regret is bounded by a specific estimation error.

$$\begin{aligned}
&\{A_t \triangleq \arg\max_{a\in[K]} X'_{ta}\hat{\beta}_t\} \\
&\Rightarrow \{X'_{tA_t}\hat{\beta}_t \geq X'_{tA_t^*}\hat{\beta}_t\} \text{ (where } A_t^* \triangleq \arg\max_{a\in[K]} X'_{ta}\beta) \\
&= \{(X_{tA_t} - X_{tA_t^*})'(\hat{\beta}_t - \beta + \beta) \geq 0\} \\
&\Rightarrow \{X'_{tA_t^*}\beta - X'_{tA_t}\beta \leq |(X_{tA_t} - X_{tA_t^*})'(\hat{\beta}_t - \beta)|\}
\end{aligned}$$

where one can identify in the left hand side of the last inequality the instantaneous regret.

Set $\gamma \in (0, 1)$ and $(\delta_1, \delta_2) \in (0, 1)^2$ such that $\delta_1 + \delta_2 < 1$ and consider that the inequalities in Lemma 2 hold for the particular sequence $(X_{tA_t} - X_{tA_t^*}; t \in [T])$ and that the set of inequalities in Lemma 4 hold.

Then, with probability $1-(\delta_1+\delta_2)$:

$$\begin{aligned}
&\textstyle\sum_{t=t_0(\delta_2,\gamma)}^{T} |(X_{tA_t} - X_{tA_t^*})(\hat{\beta}_t - \beta)| \\
&< \textstyle\sum_{t=t_0(\delta_2,\gamma)}^{T} \kappa_t' \sigma \sqrt{\frac{2\bar{\lambda}\log(T/\delta_1)}{\lambda_{\min}(M_t)}} \\
&< 2\kappa(\gamma)\sigma \sqrt{\frac{\bar{\lambda}\log(T/\delta_1)}{(1-\gamma)\underline{\lambda}}} \textstyle\sum_{t=t_0(\delta_2,\gamma)}^{T} t^{-1/2}.
\end{aligned}$$

For $t \leq t_0(\delta_2,\gamma)$, we do not have bounds over estimation errors, hence the cumulative regret is at most

$$\bar{r} t_0(\delta_2,\gamma) + 4\kappa(\gamma)\sigma \sqrt{\frac{\bar{\lambda} T \log(T/\delta_1)}{(1-\gamma)\underline{\lambda}}} \tag{12.4.1}$$

with probability $1-(\delta_1+\delta_2)$. □

In this last theorem, we showed an upper bound of order $\tilde{\mathcal{O}}(\sqrt{dT})$, since the left term is of order $d\log(T)$ and the second $\sqrt{dT\log(T)}$, the factor d being hidden within constant $\bar{\lambda}$. As such it matches the state-of-art order for high-probability regret bounds introduced in table 12.1.

In K-Arm Bandits, the differences between expected rewards directly appear in the regret bounds. The smaller the differences are, the larger the bound becomes, because it is more difficult to detect rightly the best arm. Now, because context is stochastic in our model, this optimality difference is expressed via the following margin condition.

Assumption 5. *There exists a constant m such that, for all $t \in [T]$, $(i,l) \in [K]^2, i \neq l$,*

$$\mathbb{P}_{\mathbf{X}_t}(|(X_{tl} - X_{ti})'\beta| < u) < mu. \tag{12.4.2}$$

This assumption imposes a tail condition on arm pairwise optimality difference. The larger m, the more difficult it is to pick the optimal arm at each time t.

Theorem 4. *We take assumptions 1-5. Following algorithm 14, the cumulative expected regret is at most*

$$\frac{\bar{r}d}{1-h(\gamma)^{\underline{\lambda}/\bar{\lambda}}} + \frac{C\log(T)}{(1-\gamma)} \tag{12.4.3}$$

with $C = (K-1)m8\bar{\lambda}/\underline{\lambda}(d\sigma\kappa(\gamma))^2(1+{}^3\!/_4\sqrt{\pi})$.

Theorem 4 states that following algorithm 14 leads to a expected regret of order $\mathcal{O}(d^3\log(T))$ at most. This order is coherent with Theorem 1 of Goldenshluger and Zeevi [2013] which rely on a mostly-exploiting algorithm. You see very well how the expected regret explodes with $\underline{\lambda}$ (exponent concentration on the right) and in the left term directly in the denominator. The dependence is stronger than in the high-probability result for cumulative regret in theorem 3.

12.5 Results On OLS Estimator

12.5.1 Estimation error bound

In the following lemma we present a bound on the estimation error associated to the OLS estimator $\hat{\beta}_t$.

Lemma 2. *We take assumptions 1-3. Set $\delta \in (0,1)$ and $t \in [T]$ such that M_t is invertible. For any $x \in \mathbb{R}^d$, with probability at least 1 - δ,*

$$|x'\hat{\beta}_t - x'\beta| \leq \kappa_t \sigma \sqrt{\frac{2||x||_2^2 \log(1/\delta)}{\lambda_{\min}(M_t)}}$$

with $\kappa_t = \sqrt{2 + \log\left(1 + t\bar{\lambda}/\lambda_{\min}(M_t)\right)}$.

This lemma is crucial to the proofs of the regret bounds presented in the sections above. It is quite common in the bandit literature especially in UCB-like approaches where those estimation errors help guide exploration. Below lies a proof sketch of this lemma, the full proof is in the supplementary material.

Proof sketch. We follow the steps of Lemma B.4 of Rusmevichientong and Tsitsiklis [2010] for this proof. We decompose the OLS estimate (equation 12.2.15) as:

$$\hat{\beta}_t = \beta + M_t^{-1}\xi_t \tag{12.5.1}$$

where M_t was defined in equation 12.2.16 and

$$\xi_t \triangleq \sum_{k\in[t-1]} X_k \varepsilon_k. \tag{12.5.2}$$

Fix $x \in \mathbb{R}^d$, we apply lemma 5 (in supplementary material) on self-normalized processes for $A = x'\xi_t/\sigma$ and $B = \sqrt{x'M_t x}$. The condition on (A, B) is checked is verified (see complete proof). From there, for any $y > 0$, a $1-\delta$ upper bound on $|x'\xi_t|$ is given by

$$\sigma\sqrt{\log\left(\frac{1}{\delta}\right)\left(||x||_{M_t}^2 + y\right)\left(2 + \log\left(1 + \frac{||x||_{M_t}^2}{y}\right)\right)}. \tag{12.5.3}$$

Assuming that M_t is invertible, we take $x' = \tilde{x}'M_t^{-1}$, $y = \lambda_{\min}(M_t)||x||_2^2$, which after rearranging and upper-bounding some quantities gives us the announced result. □

Lemma 2 shows how controlling the confidence ball is a matter of controlling the lowest eigenvalue of M_t. In the next subsection, we rely on a concentration result for lowest eigenvalues from matrix martingale theory.

12.5.2 Controlling lowest eigenvalue of M_t

Lemma 3 (Tropp, 2011, Theorem 3.1)**.** *Consider a finite adapted sequence $\{U_k; k \in [t]\}$ of positive semi-definite matrices of dimension d, and suppose that $\lambda_{\max}(U_k) \leq \bar{\lambda}$ almost surely. For all $\mu \geq 0, \gamma \in [0;1)$, we have:*

$$\mathbb{P}\left[\lambda_{\min}\left(\sum_{k=1}^t U_k\right) \leq (1-\gamma)\mu \text{ and } \lambda_{\min}\left(\sum_{k=1}^t \mathbb{E}_{k-1}[U_k]\right) \geq \mu\right] \leq d h(\gamma)^{\mu/\bar{\lambda}}.$$

We adapt lemma 3 to our setting in the following lemma.

Lemma 4. *We take assumptions 1-3. Set $\gamma \in (0,1)$. With probability $1-\delta$,*

$$\bigcap_{t\in\{t_0(\delta,\gamma),\dots,T\}} \left\{ \lambda_{\min}\left({}^{1}\!/_{t} M_t\right) > \frac{(1-\gamma)\underline{\lambda}}{2} \right\}. \tag{12.5.4}$$

Proof. Consider lemma 3, taking $U_k = X_k X_k'$ for all $k \in [t]$, for some fixed $t \in [T]$. First of all:

$$\begin{aligned}
&\lambda_{\min}\big(\textstyle\sum_{k=1}^{t} \mathbb{E}_{k-1}[U_k]\big) \\
&= \lambda_{\min}\big(\textstyle\sum_{k=1}^{t} \mathbb{E}\left[X_k X_k' | X_1, \dots, X_{k-1}\right]\big) \\
&> \textstyle\sum_{k=1}^{t} \lambda_{\min}\big(\mathbb{E}\left[X_k X_k' | X_1, \dots, X_{k-1}\right]\big) \text{ by concavity} \\
&> \underline{\lambda} t \text{ by definition 12.2.5.}
\end{aligned}$$

Now, setting $\mu = \underline{\lambda} t$, we get:

$$\mathbb{P}\left[\lambda_{\min}\left({}^{1}\!/_{t} M_t\right) \leq (1-\gamma)\underline{\lambda}\right] \leq d(h(\gamma))^{\underline{\lambda} t/\bar{\lambda}}.$$

Equivalently, pick t superior or equal to

$$\max\left\{ d; \frac{\bar{\lambda}\log(d/\delta_0)}{\underline{\lambda}\log(1/h(\gamma))} \right\} \tag{12.5.5}$$

and, with probability $1-\delta_0$ at least

$$\lambda_{\min}\left({}^{1}\!/_{t} M_t\right) > (1-\gamma)\underline{\lambda}. \tag{12.5.6}$$

Now, consider $S \triangleq \{d2^k : d2^k \leq T, k \in \mathbb{N}\}$. With probability at least $1-\delta$,

$$\bigcap_{\substack{t\in S \\ t\geq t_1}} \left\{ \lambda_{\min}\left({}^{1}\!/_{t} M_t\right) > (1-\gamma)\underline{\lambda} \right\} \tag{12.5.7}$$

setting $t_1 = \max\left\{ d; \dfrac{\bar{\lambda}\log(\mathbf{card}(S)d/\delta)}{\underline{\lambda}\log(1/h(\gamma))} \right\}$.

The cardinal of S is at most $\log(T/d)/\log(2)$ hence $t_1 \geq t_0(\delta,\gamma)$.

Lastly, leveraging the fact that $\{X_k X_k'; k \in [t]\}$ is a set of semi-definite positive matrices,

$$\begin{aligned}
&\forall t \in \{k \in S : k \geq t_1\} : \\
&\{\lambda_{\min}({}^{1}\!/_{t} M_t) > (1-\gamma)\underline{\lambda}\} \\
&\Rightarrow \{\lambda_{\min}({}^{1}\!/_{n} M_n) > (1-\gamma)\underline{\lambda}/2 \ ; \ \forall n \in \{t, \dots, 2t\}\}
\end{aligned}$$

which concludes the proof. □

Under Assumption 4, the convergence of $\hat{\beta}_t$ towards β is a direct consequence of lemma 4. Please note that this result hold whatever the game sequence - and hence the policy. Given the goal of the game, it makes sense to stick to a Greedy strategy.

12.6 Technical details

12.6.1 Proof section of lemma 2

Proof. We follow the steps of Lemma B.4 of Rusmevichientong and Tsitsiklis [2010] for this proof. We decompose the OLS estimate from equation 12.2.15 as:

$$\hat{\beta}_t = \beta + M_t^{-1}\xi_t \tag{12.6.1}$$

where M_t follows equation 12.2.16 and $\xi_t \overset{\Delta}{=} \sum_{k\in[t-1]} X_k\varepsilon_k$ (we play with the indexes here). We define $\mathcal{H}_k$ the following filtration, hidden to the player:

$$\forall k \in [t-1],\ \mathcal{H}_k \overset{\Delta}{=} \{X_1, \varepsilon_1, \dots, X_{k-1}, \varepsilon_{k-1}, X_k\}. \tag{12.6.2}$$

Consider the following result from de la Pena et al. [2004] on self-normalized processes.

Lemma 5 (de la Pena et al., 2004, Corollary 2.2)**.** *Let $B \geq 0$ and A be two random variables satisfying the inequality $\mathbb{E}[\exp(\gamma A - \gamma^2/2B^2)] \leq 1$ for all $\gamma \in \mathbb{R}$. Then for all $c \geq \sqrt{2}$ and $y > 0$,*

$$\mathbb{P}(|A| \geq c\sqrt{(B^2+y)(1+1/2\log(B^2/y+1))}) \leq \exp(-c^2/2). \tag{12.6.3}$$

Fix $x \in \mathbb{R}^d$, we want to apply lemma 5 for $A = x'\xi_t/\sigma$ and $B = \sqrt{x'M_tx}$. In order to apply it, we must check that

$$\forall \gamma \in \mathbb{R},\ \mathbb{E}[\exp(\gamma A - \gamma^2/2B^2)] \leq 1. \tag{12.6.4}$$

Note, for any $k \in [t]$, $A_k = \sigma^{-1}x'X_k\varepsilon_k$ and $B_k = \sqrt{x'X_kX_k'x}$.

We have that :

$$\begin{aligned}\mathbb{E}[\exp(\gamma A_k - \gamma^2/2B_k^2)|\mathcal{H}_k] &= \frac{\mathbb{E}\left[\exp\left(\gamma\sigma^{-1}x'X_k\varepsilon_k\right)\middle|\,\mathcal{H}_k\right]}{\exp\left(\gamma^2/2x'X_kX_k'x\right)} \text{ since } X_k \text{ is } \mathcal{H}_k\text{-measurable} \\ &\leq \frac{\exp\left(\gamma^2/2\ x'X_kX_k'x\right)}{\exp\left(\gamma^2/2\ x'X_kX_k'x\right)} = 1\end{aligned}$$

relying on the σ sub-Gaussian property of $\varepsilon_k|\mathcal{H}_k$. Following a recursive argument:

$$\begin{aligned}&\mathbb{E}\left[\exp\left(\gamma A - \gamma^2/2B^2\right)\right] = \mathbb{E}\left[\exp\left(\textstyle\sum_{k=1}^{t}\gamma A_k - \gamma^2/2B_k^2\right)\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\exp\left(\textstyle\sum_{k=1}^{t-1}\gamma A_k - \gamma^2/2B_k^2\right)\exp(\gamma A_t - \gamma^2/2B_t^2)\middle|\,\mathcal{H}_t\right]\right] \\ &= \mathbb{E}\left[\exp\left(\textstyle\sum_{k=1}^{t-1}\gamma A_k - \gamma^2/2B_k^2\right)\mathbb{E}\left[\exp(\gamma A_t - \gamma^2/2B_t^2)|\mathcal{H}_t\right]\right] \text{ since } \{(A_k,B_k); k\in[t-1]\} \text{ are } \mathcal{H}_t\text{-measurable} \\ &\leq \mathbb{E}\left[\exp\left(\textstyle\sum_{k=1}^{t-1}\gamma A_k - \gamma^2/2B_k^2\right)\right] \text{ because of the bound found above} \\ &\leq 1 \text{ by recursive reasoning.}\end{aligned}$$

The condition is now met, so, with probability $1-\delta$, for all $y > 0$,

$$|x'\xi_t| \leq \sigma\sqrt{2\log(1/\delta)}\sqrt{(||x||_{M_t}^2 + y)(1+1/2\log(1+||x||_{M_t}^2/y))}. \tag{12.6.5}$$

Note that $\lambda_{\min}(M_t)||x||_2^2 \leq ||x||_{M_t}^2 \leq t\bar{\lambda}||x||_2^2$. Taking $y = \lambda_{\min}(M_t)||x||_2^2$, we have that $||x||_{M_t}^2 +$

$y \leq 2||x||^2_{M_t}$ and $||x||^2_{M_t}/y \leq \dfrac{t\bar{\lambda}}{\lambda_{\min}(M_t)}$. Substituting the terms, we get:

$$|x'\xi_t| \leq \sigma\sqrt{2\log(1/\delta)}\sqrt{||x||^2_{M_t}\left(2+\log\left(1+t\bar{\lambda}/\lambda_{\min}(M_t)\right)\right)}. \tag{12.6.6}$$

Now, assume M_t is invertible and take $x' = \tilde{x}'M_t^{-1}$, then:

$$x'\xi_t = \tilde{x}'\hat{\beta}_t - \tilde{x}'\beta \text{ and } ||x||^2_{M_t} = x'M_tx = \tilde{x}'M_t^{-1}M_tM_t^{-1}\tilde{x} = ||\tilde{x}||^2_{M_t^{-1}}. \tag{12.6.7}$$

Then, for any $\tilde{x} \in \mathbb{R}^d$, with probability $1-\delta$:

$$|\tilde{x}'\hat{\beta}_t - \tilde{x}'\beta| \leq \sigma\sqrt{2\log(1/\delta)}\sqrt{||\tilde{x}||^2_{M_t^{-1}}\left(2+\log\left(1+t\bar{\lambda}/\lambda_{\min}(M_t)\right)\right)}. \tag{12.6.8}$$

The final bound is found by upper-bounding $||\tilde{x}||^2_{M_t^{-1}}$ by $||\tilde{x}||^2_2/\lambda_{\min}(M_t)$. □

12.6.2 Proof of theorem 1

Proof. Consider the bandit problem in the case where β is a scalar, i.e. $d = 1$. In this setting, notice that:

$$A_t \triangleq \arg\max_{a\in[K]}(X_{ta}\hat{\beta}_t) = \arg\max_{a\in[K]}(X_{ta}\text{sign}(\hat{\beta}_t)) \tag{12.6.9}$$

and similarly

$$A_t^* \triangleq \arg\max_{a\in[K]}(X_{ta}\beta) = \arg\max_{a\in[K]}(X_{ta}\text{sign}(\beta)). \tag{12.6.10}$$

As such:

$$\begin{aligned}\textbf{regret}(\mathcal{F}_T) \triangleq \sum_{t=1}^{T}\max_a X_{ta}\beta - X_{tA_t}\beta &= \beta\sum_{t=1}^{T}\max_a(X_{ta}\text{sign}(\beta)) - \max_a(X_{ta}\text{sign}(\hat{\beta}_t)) \\ &< |\beta|\bar{r}'\sum_{t=1}^{T}\mathbb{1}\{\text{sign}(\hat{\beta}_t) \neq \text{sign}(\beta)\}\end{aligned}$$

where $\bar{r}' \triangleq \max_{a,\tilde{a}}|X_{1a} - X_{1\tilde{a}}|$. Please note that $\beta\bar{r}' = \bar{r}$ for $d = 1$. Applying lemmas 2 and 4 jointly we have with probability $1-(\delta_1+\delta_2)$: $\forall t \in (t_0(\delta_2,\gamma), T]$

$$\{\text{sign}(\hat{\beta}_t) \neq \text{sign}(\beta)\} \Rightarrow \left\{|\beta| < \kappa_t'\sigma\sqrt{\frac{2\bar{\lambda}\log(1/\delta_1)}{\lambda_{\min}(M_t)}}\right\} \Rightarrow \left\{|\beta| < t^{-1/2}\kappa(\gamma)\sigma\sqrt{\frac{2\bar{\lambda}\log(1/\delta_1)}{\underline{\lambda}(1-\gamma)/2}}\right\} = \{t < t_1\}$$

where $t_1 = \dfrac{(2\kappa(\gamma)\sigma)^2}{\beta^2}\dfrac{\bar{\lambda}\log(1/\delta_1)}{\underline{\lambda}(1-\gamma)}$. From there, with probability $1-(\delta_1+\delta_2)$:

$$\sum_{t=1}^{T}\mathbb{1}\{\text{sign}(\hat{\beta}_t) \neq \text{sign}(\beta)\} < t_0(\delta_2,\gamma) + \sum_{t=t_0(\delta_2,\gamma)}^{T}\mathbb{1}\{t < t_1\} < \max\{t_0(\delta_2,\gamma), t_1\}$$

which concludes the proof. □

12.6.3 Proof of theorem 2

Proof. Focusing on the one-dimensional case, the expected regret is rewritten:

$$\sum_{t=1}^{T} \mathbb{E}_{(\mathbf{X}_1,\dots,\mathbf{X}_t)} \left[\max_a X_{ta}\beta - X_{tA_t}\beta \right]. \tag{12.6.11}$$

Since one only needs to known the sign of β in order to find the optimal arm, we have

$$\begin{aligned}
&\mathbb{E}_{(\mathbf{X}_1,\dots,\mathbf{X}_t)} \left[\max_a X_{ta}\beta - X_{tA_t}\beta \right] \\
&= \mathbb{E}_{(\mathbf{X}_1,\dots,\mathbf{X}_t)} \left[\max_a X_{ta}\beta - X_{tA_t}\beta \middle| \operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t) \right] \mathbb{P}_{(\mathbf{X}_1,\dots,\mathbf{X}_t)} \left[\operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t) \right] \\
&= \mathbb{E}_{\mathbf{X}_t} \left[\max_a X_{ta}\beta - X_{tA_t}\beta \middle| \operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t) \right] \mathbb{P}_{(\mathbf{X}_1,\dots,\mathbf{X}_{t-1})} \left[\operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t) \right]
\end{aligned}$$

where the last equality stands from the fact that $(X_{tA_t^*}, X_{tA_t})$ are independent of $(\mathbf{X}_1, \dots, \mathbf{X}_{t-1})$ conditionally to $\mathbf{X}_t$ and $\hat{\beta}_t$ is independent of $\mathbf{X}_t$ conditionally to $(\mathbf{X}_1, \dots, \mathbf{X}_{t-1})$. The first term is bounded by $\tilde{r}$. Now, applying jointly lemmas 2 and 4, we have that with probability $1 - (\delta_1 + \delta_2)$:

$$\forall t \in (t_0(\delta_2, \gamma); T], \{\operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t)\} \Rightarrow \left\{ |\beta| < 2\kappa(\gamma)\sigma \sqrt{\frac{\bar{\lambda} \log(1/\delta_1)}{t\underline{\lambda}(1-\gamma)}} \right\} = \{t < t_1\} \tag{12.6.12}$$

where $t_1 = \dfrac{(2\kappa(\gamma)\sigma)^2}{\beta^2} \dfrac{\bar{\lambda} \log(1/\delta_1)}{\underline{\lambda}(1-\gamma)}$. As such:

$$\begin{aligned}
\sum_{t=1}^{T} \mathbb{P}_{(\mathbf{X}_1,\dots,\mathbf{X}_{t-1})} \left[\operatorname{sign}(\beta) \neq \operatorname{sign}(\hat{\beta}_t) \right] &< (\delta_1 + \delta_2)(T - \max\{t_0(\delta_2, \gamma); t_1\})_+ + \max\{t_0(\delta_2, \gamma); t_1\} \\
&< 2 + \max \left\{ t_0\left(1/T, \gamma\right); \frac{(2\kappa(\gamma)\sigma)^2}{\beta^2} \frac{\bar{\lambda} \log(T)}{\underline{\lambda}(1-\gamma)} \right\}
\end{aligned}$$

where the last inequality comes from renormalizing probability risks (δ_1, δ_2) by $1/T$. Rearranging the terms provide the announced bound. □

12.6.4 Proof of theorem 4

Proof. Consider the instantaneous regret at time t,

$$\mathbf{regret}(\mathcal{F}_t; t) \triangleq \max_{a \in [K]} X'_{ta}\beta - X'_{tA_t}\beta.$$

In the interest of space, we will write in this proof $\mathbf{X}_{t^-}$ for the set $(\mathbf{X}_1, \dots, \mathbf{X}_t)$. In the following we aim to control the expected instantaneous regret

$$\mathbb{E}_{\mathbf{X}_{t^-}} [\mathbf{regret}(\mathcal{F}_t; t) | \mathrm{E}], \tag{12.6.13}$$

when we have control over the lowest eigenvalue of M_t:

$$\mathrm{E} \triangleq \{\lambda_{\min}(M_t) > t\underline{\lambda}(1-\gamma)\}. \tag{12.6.14}$$

To start we will condition this expectation on the optimal arm and to do so we define the following pseudo-partition of support space $\mathcal{X}$:

$$\forall i \in [K],\ \mathcal{R}_i \triangleq \{x \in \mathcal{X} : x'_{i.}\beta > x'_{j.}\beta; \forall j \neq i\}.$$

Now, let us to condition the expected regret:

$$\begin{aligned}
\mathbb{E}_{\mathbf{X}_{t^-}}[\mathbf{regret}(\mathcal{F}_t;t)|\mathrm{E}] &= \sum_{i=1}^{K} \mathbb{E}_{\mathbf{X}_{t^-}}[\mathbf{regret}(\mathcal{F}_t;t)|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_i]\mathbb{P}(\mathbf{X}_t \in \mathcal{R}_i) \\
&\leq \sum_{i=1}^{K} \mathbb{E}_{\mathbf{X}_{t^-}}[1\{\max_a X'_{ta}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\}\mathbf{regret}(\mathcal{F}_t;t)|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_i]\mathbb{P}(\mathbf{X}_t \in \mathcal{R}_i) \\
&\leq \sum_{i=1}^{K}\sum_{l\neq i} \mathbb{E}_{\mathbf{X}_{t^-}}[1\{X'_{tl}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\}(X_{ti} - X_{tl})'\beta|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_i]\mathbb{P}(\mathbf{X}_t \in \mathcal{R}_i)
\end{aligned} \tag{12.6.15}$$

where the first equality comes from law of total probability, the first inequality is due to the fact the fact that regret is a non-negative term and for i fixed we conditioned on the fact that i is optimal. The last inequality comes also from law of total probability - some probabilities were put away.

From now, fix $(i,l) \in [K]^2, i \neq l$ and focus on the expectation within the last inequality. We are going to partition the random variable $(X_{ti} - X_{tl})'\beta$. Consider the intervals

$$\forall h \in \mathbb{N},\ I(h) \triangleq \{x \in \mathcal{X} : (X_{ti} - X_{tl})'\beta \in (Ch; C(h+1)]\}$$

with constant C remaining to be chosen. Then:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{X}_{t^-}}\left[1\{X'_{tl}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\}(X_{ti} - X_{tl})'\beta\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l\right] \\
&= \sum_{h\in\mathbb{N}} \mathbb{E}_{\mathbf{X}_{t^-}}\left[1\{X'_{tl}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\}(X_{ti} - X_{tl})'\beta\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right]\mathbb{P}[\mathbf{X}_t \in I(h)] \\
&\leq \sum_{h\in\mathbb{N}} C(h+1)\mathbb{P}_{\mathbf{X}_{t^-}}\left[X'_{tl}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right]\mathbb{P}[\mathbf{X}_t \in I(h)] \\
&\leq \sum_{h\in\mathbb{N}} mC^2(h+1)^2\mathbb{P}_{\mathbf{X}_{t^-}}\left[X'_{tl}\hat{\beta}_t \geq X'_{ti}\hat{\beta}_t\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right]
\end{aligned} \tag{12.6.16}$$

where the first inequality comes from taking the upper bound of interval $I(h)$ for $(X_{ti} - X_{tl})'\beta$ and the second inequality comes from Assumption 5.

From there, notice the following:

$$\begin{aligned}
\mathbb{P}_{\mathbf{X}_{t^-}}\left[X'_{ti}\hat{\beta}_t \geq X'_{tl}\hat{\beta}_t\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right] &= \mathbb{P}_{\mathbf{X}_{t^-}}\left[(X_{tl} - X_{ti})'(\hat{\beta}_t - \beta + \beta) \leq 0\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right] \\
&\leq \mathbb{P}_{\mathbf{X}_{t^-}}\left[(X_{tl} - X_{ti})'(\hat{\beta}_t - \beta) \leq -Ch\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right] \\
&\leq \mathbb{P}_{\mathbf{X}_{t-1^-}}\left[|\hat{\beta}_t - \beta| \geq Ch/(2\sqrt{d\bar{\lambda}})\middle|\mathrm{E}, \mathbf{X}_t \in \mathcal{R}_l \cap I(h)\right] \\
&\leq \exp\left(-\frac{(Ch)^2 t\underline{\lambda}(1-\gamma)}{8\bar{\lambda}d^2\kappa^2\sigma^2}\right)
\end{aligned}$$

where the last inequality stands from lemma 2 and cashing in event E. Taking $C^2 = \frac{8\bar{\lambda}d^2\kappa^2\sigma^2}{t\underline{\lambda}(1-\gamma)}$ and plugging (12.6.16) into (12.6.15) renders:

$$\mathbb{E}_{\mathbf{X}_{t^-}}[\mathbf{regret}(\mathcal{F}_t;t)|\lambda_{\min}(M_t) > t\underline{\lambda}(1-\gamma)] \leq \sum_{i\in[K]}\left(\sum_{l\in[K]\setminus\{i\}}\sum_{h\in\mathbb{N}} mC^2(h+1)^2\exp(-h^2)\right) \leq C'/t$$

with $C' = (K-1)m8\bar{\lambda}d^2\kappa^2\sigma^2(1+3/4\sqrt{\pi})/(\underline{\lambda}(1-\gamma))$. This last inequality comes from the following inequality:

$$\sum_{h\in\mathbb{N}}(h+1)^2\exp(-h^2) \leq \int_{h=0}^{+\infty}(h+1)^2\exp(-h^2)dh = 1+3/4\sqrt{\pi} \tag{12.6.17}$$

with integration by parts. Those results are all conditional to the control over $\lambda_{\min}(M_t)$. Let $\bar{\mathrm{E}}$ denote the complementary event to E, we have:

$$\begin{aligned}\sum_{t\in[T]}\mathbb{E}_{\mathbf{X}_{t^-}}[\mathbf{regret}(\mathcal{F}_t;t)] &\leq \sum_{t\in[T]}\mathbb{E}_{\mathbf{X}_{t^-}}[\mathbf{regret}(\mathcal{F}_t;t)|\mathrm{E}] + \bar{r}\ \mathbb{P}(\bar{E}|\mathcal{F}_t)\\ &\leq \sum_{t\in[T]}\frac{C'}{t} + \bar{r}\sum_{t\in[T]}\mathbb{P}(\bar{E}|\mathcal{F}_t)\\ &\leq C'\log(T) + \frac{\bar{r}d}{1-h(\gamma)^{\underline{\lambda}/\bar{\lambda}}}.\end{aligned}$$

This gives us the desired bound. □

12.7 Conclusion

In this work, we study a simple greedy algorithm for linear contextual bandits, under a realistic assumption of context distribution. We showed that cumulative pseudo regret is upper bounded by $\tilde{\mathcal{O}}(\sqrt{dT})$ and cumulative expected pseudo regret is upper bounded by $\mathcal{O}(d^3\log(T))$, where d is the context dimension and T the number of rounds. Those results match state-of-art results in terms of bound order. They were already established for greedy policies under specific instances of the linear contextual bandit problem by Bastani et al. [2017] and Kannan et al. [2018]; we extended their results to the generic problem.

Bound dependency on K The upper bound for cumulative regret for SUPLINUCB from Chu et al. [2011] has a $\log^3$ dependency in K.

Although the number of arms K has not naturally appeared in our bounds, it is hidden within $\underline{\lambda}$. Although an exact relationship is difficult to explicit, it is clear that augmenting K would tend to diminish $\underline{\lambda}$. Indeed, if we were to take $\mathbf{X}_t = \mathbb{R}^d$, we would have $\underline{\lambda} = 0$ from the way it is defined: only one point would be optimal and so a lower bound would not be properly set.

Non-stationary setting Although distribution $\mathcal{P}$ is not indexed over time or game history, we can assume an adversarial setting as long as all the assumptions we made hold throughout the game. For unbounded distributions, we can modify the definition of $\bar{r}$ and update regret proofs quite easily.

Improving the $\underline{\lambda}$ constants In theorem 3, we can improve the second term of the bound. Following algorithm 14, the lower bound on $\lambda_{\min}(\mathbb{E}[X_t X_t'])$ is given by $\underline{\lambda}$ for all $t < t_0(\delta_2, \gamma)$. Now, since we apply lemma 2 for all $t \in \{t_0(\delta_2, \gamma), \dots, T\}$, we have that a lower bound on $\lambda_{\min}(\mathbb{E}[X_t X_t'])$ is given by

$$\underline{\lambda}' \triangleq \inf_{\hat{\beta} \in \mathcal{B}} \lambda_{\min} \left(\mathbb{E}\left[X_1 X_1' \middle| \hat{\beta} \right] \right) \tag{12.7.1}$$

where

$$\mathcal{B} \triangleq \left\{ b \in \mathbb{R}^d : ||b - \beta||_1 < \kappa(\gamma)\sigma \sqrt{\frac{2d \log(1/\delta)}{t_0(\delta_2, \gamma)\underline{\lambda}}} \right\}. \tag{12.7.2}$$

Since $\mathcal{B} \subset \mathbb{R}^d$, $\underline{\lambda} \leq \underline{\lambda}'$ by their definitions. As such, the regret bound from theorem 3 becomes:

$$\bar{r} t_0(\delta_2, \gamma) + 4\kappa(\gamma)\sigma \sqrt{\frac{\bar{\lambda} T \log(T/\delta_1)}{(1-\gamma)\underline{\lambda}'}} \tag{12.7.3}$$

since one can show that

$$\sum_{t=t_0(\delta_2,\gamma)}^{T} (t_0(\delta_2, \gamma)\underline{\lambda} + (t - t_0(\delta_2, \gamma))\underline{\lambda}')^{-1/2} < 2\sqrt{T/\underline{\lambda}'}. \tag{12.7.4}$$

Conclusion

In this part, we examined different decision processes, where the system we are interacting with has enough stochasticity that enforces exploration in different ways: stochastic transitions, constrained actions, specific assumptions on reward function between actions.

In the first application, we looked at constrained actions: whatever the action believed to be the best, it will be sometimes unavailable and we will turn to the second best. This one might in turn also be unavailable and so on. This process forces us to draw new actions (think about your favorite restaurant that is closed, then you might try another one!)

In the second, we looked at constrained dynamics: even if we can direct the object in a given direction, the system is a bit shaky and might land in different spots. As such, for a certain range of stochastic dynamics, we are able to direct the object significantly, but also try new spots without the explicit intent of doing so.

In the third and last application, we impose a key assumption on the relation between the set of actions and the reward function encodes the passing of information through different actions. Basically, whatever the arm drawn, we will learn on the reward function associated to every other arm. In that setting, we recover proper theoretical regret bounds matching the order of UCB-analog bounds (which focuses exactly on the exploration-exploitation dilemma).

As a consequence in general, this enforced exploration, when sufficiently high, is sufficient to ensure that our behavior, focusing on exploitation does not hinder the process, looking at classic regret metric for example. Let us give more details on the three model specifications which allowed proper exploration.

Global conclusion

Contributions

Dans cette thèse, nous avons étudié plusieurs applications où la prise de décision, qu'elle soit unique ou séquentielle, est au cœur du problème.

Les trois premières applications ont été conçues comme un cadre d'apprentissage supervisé orienté tâche, qui étend le cadre classique d'apprentissage supervisé Hastie et al. [2009] au cas où une fonction de perte est définie explicitement par l'utilisateur, quantifiant la capacité à prédire la cible Y comme $\hat{Y}$ dans un contexte connu X et un contexte inconnu Z, par exemple la cible est un niveau de demande, X est l'information temporelle (jour de la semaine, saison), Z est le prix stochastique inconnu au moment de la prévision. Cette approche rapproche la prévision de la présciption et d'un cadre décisionnel. Dans la littérature, nous avons trouvé plusieurs travaux se concentrant sur la façon d'entraîner correctement un module de prévision qui vient en entrée d'un module d'optimisation, typiquement un problème complexe de prise de décision. Dans Bertsimas and Kallus [2020], les auteurs se concentrent sur la minimisation de la perte attendue conditionnelle, la perte étant définie en fonction du problème. Les auteurs ont démontré la variabilité des applications dans les problèmes de recherche opérationnelle. Dans l'article Huang et al. [2019], les auteurs tentent de modifier le résultat de la prédiction afin d'obtenir une performance globale appropriée sous une fonction de perte personnalisée. Dans Donti et al. [2017], les auteurs démontrent sur trois problèmes de RO que les réseaux neuronaux peuvent être optimisés sans problème, tant que le problème d'optimisation suivant a une perte convexe. Dans Carriere and Kariniotakis [2019], les auteurs optimisent conjointement leurs modules de prévision et d'optimisation. Ce sujet prend de plus en plus d'importance à mesure que la communauté et ses utilisateurs se rendent compte que la prévision est une chose, agir sur elle en est une autre. Il existe de nombreuses façons d'aborder la question, les méthodes que nous avons proposées sont assez légères computationnellement.

Notez que ce cadre est évidemment lié à la prise de décision séquentielle, puisqu'il équivaut à se concentrer sur la première étape du problème ; c'est pourquoi une approche pour résoudre le problème consiste en fait à utiliser un algorithme d'apprentissage par renforcement et à optimiser la politique à une étape, qui à un état X attribue le meilleur prédicteur $\hat{Y}$. Il est intéressant de noter que la modification des arbres de décision a permis d'obtenir d'assez bonnes performances lors de nos tests, avec un net avantage en termes de calcul.

Les autres applications ont été abordées avec des approches d'apprentissage par renforcement basées sur la valeur Sutton and Barto [2018], la contribution essentielle étant de montrer comment formuler ces problèmes correctement pour trouver des politiques et analyser ces résultats. En effet, les applications des chapitres 6 et 8 sont relativement simples car il s'agit de processus simples. Cependant, le questionnaire intelligent (chapitre 9) et le problème de gestion de l'insuline (chapitre 7) étaient des applications beaucoup plus étendues, nécessitant beaucoup plus de travail sur la modélisation elle-même. Dans le questionnaire intelligent, nous avons essayé de construire une version intelligente et continue de l'arbre de décision classique Breiman et al. [1984], avec la tâche formulée comme posant séquentiellement la question la plus pertinente pour la prédiction de la cible sur la base d'informations incomplètes. Ce projet serait très utile lorsque nous souhaitons prédire une variable cible, mais que le coût de récupération des données n'est pas négligeable, ce qui conduit à des compromis, comme par exemple l'interaction de l'utilisateur sur un site web. L'arbre de décision est une base de référence naturelle, ainsi que le meilleur sous-ensemble fixe de variables. Il existe plusieurs approches basées sur des règles, soit ancrées dans la connaissance du domaine, voir [Dunlop, 2019, Nokelainen et al., 2001], soit basées sur l'exploration de données,

voir [Mwamikazi et al., 2014]. Plus récemment, [Chen et al., 2018] s'est attaqué au problème des 20 questions et [Besson et al., 2018] à un problème d'évaluation de diagnostic de santé, tous deux utilisant une approche d'apprentissage par renforcement.

Dans le problème de la gestion de l'insuline, nous avons axé notre application sur les patients atteints de diabète de type I qui ne sont pas équipés d'une pompe à insuline et s'auto-injectent donc de l'insuline avant les repas pour contrôler leur taux de glycémie. Il y a eu beaucoup de travaux autour du sujet de la gestion de l'insuline, beaucoup se concentrant sur le pancréas artificiel. Plus précisément, de nombreuses études se sont appuyées sur les méthodes PID (proportionnelle, intégrative, dérivée), MPC (Model predictive control), contrôle optimal et techniques floues. Les approches d'apprentissage par renforcement (RL) sont apparues progressivement au début des années 2000 et plus encore dans la décennie suivante, avec des cas d'utilisation différents. Les auteurs de Ngo et al. [2018] ont par exemple étudié par des méthodes RL la politique optimale d'injection d'insuline, c'est-à-dire comment délivrer au mieux l'insuline à l'organisme afin d'avoir le meilleur impact possible sur la glycémie. Cette étude, basée sur le modèle cinétique insuline-glycose, a permis de mieux comprendre comment concevoir un médicament. Dans l'article Javad et al. [2019], les auteurs appliquent l'algorithme Q-Learning sur une cohorte, dans le but de trouver une gamme appropriée de doses d'insuline à prescrire. Ils ont cherché à personnaliser le traitement en représentant chaque patient par des informations pertinentes : hémoglobine glyquée passée, IMC, niveau d'activité, consommation d'alcool, toutes discrétisées. La récompense est ensuite exprimée en fonction de l'évolution de l'hémoglobine glyquée. On peut noter que la personnalisation est bien sûr limitée à la définition de l'état. De plus, de telles études sont difficiles à mener pour tout chercheur, car elles sont réalisées sur des données réelles de patients et non sur des données simulées. En se basant sur un simulateur complexe de diabète, les auteurs de Daskalaki et al. [2013] ont proposé un double algorithme Acteur-Critique dans le cas de la surveillance continue du glucose. Ils ont contrôlé et mis à jour simultanément le débit de base et le ratio glucides/insuline, en s'appuyant sur le conseiller bolus standard pour prescrire l'insuline en bolus. Une étude plus récente de Sun et al. [2018], basée sur le même simulateur, a examiné la mise à jour du ratio glucides/insuline entre les repas plutôt que quotidiennement, ainsi que la robustesse de la politique à l'incertitude concernant les repas sautés, la taille et l'heure des repas. Notre approche a consisté à utiliser l'apprentissage par renforcement conjointement avec un simulateur de l'interaction gluco-insuline pour différents patients virtuels et pour des plans de repas stricts. Nous avons ensuite pris comme paramètre de la politique le plan de repas suivi, afin de trouver le meilleur plan de repas en termes de variations de la glycémie.

Dans la dernière section, nous avons exploré des processus spécifiques qui ont été construits de telle sorte que le dilemme exploitation-exploration Sutton and Barto [2018] est en quelque sorte résolu : quelle que soit la décision prise, nous sommes assurés que l'exploration a lieu et que l'on peut donc suivre une stratégie avide Bastani et al. [2017]. La connaissance de ce que l'on étudie (approche basée sur un modèle, approche bayésienne, etc.) est toujours un avantage du point de vue de l'information, même si elle est parfois plus difficile à modéliser, à résoudre numériquement/exactement, etc. Cela nous permet de dire que ce que nous savons du système est complété par les données d'une certaine manière. Dans nos expériences, nous avons analysé le comportement de systèmes assez particuliers, qui ne nécessitent pas que l'agent se préoccupe du problème d'exploration-exploitation : quoi que fasse l'agent, tout se passera bien, pour diverses raisons, parmi lesquelles des transitions très stochastiques. Ce type de connaissance a priori pourrait être exploité pour réguler intelligemment le niveau d'exploration appliqué.

Outre la connaissance du système, il y a plusieurs conditions à remplir pour appliquer une approche d'apprentissage supervisé ou d'apprentissage par renforcement axée sur les tâches, que nous présentons ci-dessous. Ensuite, nous discutons des avantages de la méthode ainsi que de leurs risques inhérents, notamment lors du déploiement.

Conditions

Il y a deux exigences de base pour que ces approches (apprentissage supervisé orienté tâche et apprentissage par renforcement) fonctionnent.

Premièrement, l'état et l'action doivent être très clairement définis : l'action appartenant à l'ensemble des événements actionnables imposés par l'agent au système et l'état étant l'ensemble des descripteurs du système, et supposément les informations suffisantes pour prendre des décisions optimales. Le cadre temporel doit également être très clair. Dans la première partie de cette thèse, nous devons définir correctement X, Y, Z et la fonction de perte, qui est l'équivalent dans le cadre de la prise de décision en une étape.

Deuxièmement, l'idéal est de disposer d'un simulateur de données qui permet d'exécuter différentes actions à volonté ; les données d'observation peuvent être un gros piège si les actions entreprises dépendent de caractéristiques observées, mais pas nécessairement rapportées. Pour la première partie de cette thèse, si la perte n'est pas connue explicitement, nous avons besoin soit d'un simulateur, soit de données d'observation avec une exploration aléatoire appropriée.

Maintenant, il y a le problème permanent de la source de données, lié à l'hypothèse sous-jacente d'indépendance et de distribution identique (i.i.d.). Les données d'observation avec des actions dépendantes des données sont un exemple typique. Le problème qui se pose est de décider quelles données sont fournies en entrée : disons que vous avez généré des données pour N patients différents, vous avez trois grandes options : soit combiner toutes les données, et certaines caractéristiques individuelles en tant que caractéristiques, soit le faire uniquement pour des patients relativement similaires (le regroupement vient à l'esprit), soit appliquer une approche individualisée. On peut vouloir essayer plusieurs possibilités, tant que la quantité de données est suffisamment élevée pour permettre de construire des estimations.

Avantages de ces méthodes

Le premier avantage évident est la polyvalence des applications qui peuvent être atteintes avec des approches raisonnablement similaires. Il s'agit en effet de cadres assez directs pour la prise de décisions (simples).

Ces approches sont également très flexibles :

- Dans l'apprentissage supervisé, les méthodes sont assez performantes et flexibles par rapport à la taille des données d'entrée.

- Dans de nombreuses applications, nous avons dû instancier nous-mêmes un simulateur ou nous appuyer sur un simulateur existant ; on peut donc souhaiter réutiliser tout cela mais en ajoutant certains éléments dans le vecteur d'état ou autre ; le cadre est suffisamment flexible pour que rien ne change ; maintenant, bien sûr, on peut avoir à simuler ou à observer un peu plus de données en fonction du nombre de dimensions ajoutées.

Ces méthodes, en plus de fournir des politiques performantes, donnent un aperçu de ce qu'est une politique performante, dans le cadre de la fonction récompenses/pertes. On l'a vu par exemple avec le développement d'AlphaGo Silver et al. [2018, 2017] qui, entre autres, a développé des stratégies inconnues des meilleurs joueurs de Go.

Il y a aussi des développements inattendus. Par exemple, sortir des sentiers battus est payant dans les problèmes de RL : dans le cas du diabète, on passe avant le problème de la quantité d'insuline à prendre, mais on considère d'abord le moment où la personne prend ses repas, ce qui peut être soit préjudiciable, soit plutôt utile pour équilibrer naturellement la glycémie. Il est intéressant de noter qu'aux échecs, les variantes du jeu sont explorées à l'aide de l'apprentissage par renforcement, afin d'accélérer la recherche de nouvelles variantes créatives, voir Tomašev et al. [2020].

Notez également l'avantage évident de disposer d'un simulateur réaliste construit dans un domaine où les expériences réelles sont contraires à l'éthique, trop coûteuses, trop longues, etc. et qui nous permet de poser des questions qui n'auraient pas été abordées autrement et à des coûts opérationnels nuls : trouver un plan de repas régulier qui gère bien la dynamique de la glycémie par lui-même, qui peut s'adapter aux repas manqués, aux changements d'activité, etc.

En pratique

Énumérons quelques vérifications classiques lorsqu'un modèle est destiné à être utilisé dans la vie réelle :

- (Basic) Vérifier la validité de l'entrée (est-elle similaire aux entrées précédentes, n'y a-t-il pas de valeurs anormalement grandes/faibles, y a-t-il des valeurs manquantes que nous n'avions pas prévu de traiter, etc.)
- (Basic) Vérifier la validité de la sortie (si la sortie est une probabilité, vérifiez la plage [0,1], comparez avec les versions précédentes de la politique, comparez également avec des lignes de base simples)
- Non-stationnarité de la dynamique, ce qui rendrait le modèle antérieur sous-optimal et potentiellement complètement erroné : on pourrait mettre à jour le modèle de temps en temps comme cela a été fait dans Garivier and Moulines [2011], où les auteurs proposent une UCB actualisée et une UCB à fenêtre glissante. Voir aussi Lecarpentier and Rachelson [2020] où les auteurs proposent un algorithme pour obtenir des politiques robustes aux changements dans un processus de décision de Markov.
- Changements soudains : systèmes d'alarme, détection des points de changement et tout (lié aux deux premiers points bien sûr). Dans Lebarbier [2005], l'auteur étudie le problème de la détection des points de changement dans la moyenne d'un signal corrompu par un bruit gaussien additif sans aucune hypothèse sur le nombre de changements ou leur position. Pour les revues, on peut s'intéresser à Lai [1995], Aminikhanghahi and Cook [2017] ou Truong et al. [2020]. On peut noter le grand nombre de techniques rapportées. Notez que le problème est parfois encadré hors ligne mais qu'il peut généralement être adapté au cadre en ligne ; voir par exemple Chen et al. [2020], Tsechpenakis et al. [2006], Harchaoui et al. [2009].
- est le temps d'acquisition en quelque sorte prédictif ? vérifier par exemple avec l'importance de la variable conditionnelle donnée par les forêts aléatoires par exemple, Strobl et al. [2008].

- Apprentissage par renforcement robuste et sécurisé (eng: Safe Reinforcement Learning), notez qu'en optimisant simplement la somme des récompenses, vous n'obtenez pas nécessairement une bonne politique à la fin ; voir Berkenkamp et al. [2017], Cowen-Rivers et al. [2020], Derman et al. [2020], Garcıa and Fernández [2015].

Ce thème est particulièrement bien adapté à 2020, car de nombreuses applications basées sur l'historicité ont mal fonctionné, voir Heaven [2020], McNellis [2020]. Une expérience amusante de l'artiste Simon Weckert a été réalisée une fois à Berlin, avec un chariot de 100 téléphones sur la route. Le service Google Maps a prédit un embouteillage en conséquence Weckert [2020].

Les algorithmes online ne sont pas facilement gérables, car on passe de l'absence de données et d'une connaissance limitée à :

- Des données non-iid (puisqu'elles ont été sélectionnées séquentiellement) attention aux données d'observation, qui requièrent une attention particulière, voir par exemple dans Gottesman et al. [2018] comment les auteurs évaluent les algorithmes RL de manière impartiale et dans Lu et al. [2018] comment les auteurs corrigent le problème des données d'observation de telle sorte que l'algorithme RL construit ne soit pas tellement biaisé par la manière dont les données ont été collectées

- Impossible de s'arrêter à mi-chemin pour pauser et essayer d'autres approches

L'approche classique de la prise de décision multi-étapes se concentre sur la somme cumulée des récompenses, mais de nombreuses autres statistiques peuvent être pertinentes et différentes fonctions de récompense peuvent être intéressantes (approche multi-objectifs) car, en fin de compte, elle définit fondamentalement ce qu'est une bonne politique ; tout comme la fonction de perte décrit un bon prédicteur. Voir par exemple Van Moffaert and Nowé [2014], Yang et al. [2019].

Il peut être utile de faire intervenir des experts du domaine pour qu'ils votent en faveur d'une politique ou d'une autre, voir la référence sur l'apprentissage par renforcement inverse Ng et al. [2000].

Bibliography

Bibliography

Abbasi-Yadkori, Y., Pál, D., and Szepesvári, C. (2011). Improved algorithms for linear stochastic bandits. In Advances in Neural Information Processing Systems, pages 2312–2320.

Aglin, G., Nijssen, S., and Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. In AAAI, pages 3146–3153.

Agrawal, S. and Goyal, N. (2013). Thompson sampling for contextual bandits with linear payoffs. In International Conference on Machine Learning, pages 127–135.

Aminikhanghahi, S. and Cook, D. J. (2017). A survey of methods for time series change point detection. Knowledge and information systems, 51(2):339–367.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. Machine learning, 47(2-3):235–256.

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C., and Freitas, A. A. (2013). Automatic design of decision-tree algorithms with evolutionary algorithms. Evolutionary computation, 21(4):659–684.

Bastani, H., Bayati, M., and Khosravi, K. (2017). Mostly exploration-free algorithms for contextual bandits. arXiv preprint arXiv:1704.09011.

Bennett, K. (1995). Global tree optimization: A non-greedy decision tree algorithm. Computing Sciences and Statistics, 26.

Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In Advances in neural information processing systems, pages 908–918.

Bertrand, G. and Papavasiliou, A. (2020). Adaptive trading in continuous intraday electricity markets for a storage unit. IEEE Transactions on Power Systems, 35(3):2339–2350.

Bertsekas, D., Tsitsiklis, J., and Τσιτσικλής, G. (1996). Neuro-dynamic Programming. Anthropological Field Studies. Athena Scientific.

Bertsimas, D. and Kallus, N. (2020). From predictive to prescriptive analytics. Management Science, 66(3):1025–1044.

Besson, R., Pennec, E. L., Allassonniere, S., Stirnemann, J., Spaggiari, E., and Neuraz, A. (2018). A model-based reinforcement learning approach for a rare disease diagnostic task. arXiv preprint arXiv:1811.10112.

Bothe, M. K., Dickens, L., Reichel, K., Tellmann, A., Ellger, B., Westphal, M., and Faisal, A. A. (2013). The use of reinforcement learning algorithms to meet the challenges of an artificial pancreas. Expert review of medical devices, 10(5):661–673.

Boukas, I., Ernst, D., Théate, T., Bolland, A., Huynen, A., Buchwald, M., Wynants, C., and Cornélusse, B. (2020). A deep reinforcement learning framework for continuous intraday market bidding. arXiv preprint arXiv:2004.05940.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). Classification and regression trees. CRC press.

Carpentier, P., Chancelier, J.-P., Cohen, G., and De Lara, M. (2015). Stochastic Multi-Stage Optimization: At the Crossroads between Discrete Time Stochastic Control and Stochastic Programming, volume 75. Springer.

Carriere, T. and Kariniotakis, G. (2019). An integrated approach for value-oriented energy forecasting and data-driven decision-making application to renewable energy trading. IEEE Transactions on Smart Grid, 10(6):6933–6944.

Chen, Y., Chen, B., Duan, X., Lou, J.-G., Wang, Y., Zhu, W., and Cao, Y. (2018). Learning-to-ask: Knowledge acquisition via 20 questions. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1216–1225.

Chen, Y., Wang, T., and Samworth, R. J. (2020). High-dimensional, multiscale online changepoint detection. arXiv preprint arXiv:2003.03668.

Chou, K.-C., Lin, H.-T., Chiang, C.-K., and Lu, C.-J. (2015). Pseudo-reward algorithms for contextual bandits with linear payoff functions. In Asian Conference on Machine Learning, pages 344–359.

Chu, W., Li, L., Reyzin, L., and Schapire, R. (2011). Contextual bandits with linear payoff functions. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 208–214.

Colbaugh, R., Glass, K., Tremblay, M., and Rudolf, C. (2018). Learning to identify rare disease patients from electronic health records. AMIA Annu Symp Proc.

Corchero, C., Heredia, F. ., and Mijangos, E. (2011). Efficient solution of optimal multimarket electricity bid models. In 2011 8th International Conference on the European Energy Market (EEM), pages 244–249.

Cowen-Rivers, A. I., Palenicek, D., Moens, V., Abdullah, M., Sootla, A., Wang, J., and Ammar, H. (2020). Samba: Safe model-based & active reinforcement learning.

Daskalaki, E., Diem, P., and Mougiakakou, S. G. (2013). An actor–critic based controller for glucose regulation in type 1 diabetes. Computer methods and programs in biomedicine, 109(2):116–125.

De Cock, D. (2011). Ames, iowa: Alternative to the boston housing data as an end of semester regression project. Journal of Statistics Education, 19(3).

de la Pena, V. H., Klass, M. J., and Lai, T. L. (2004). Self-normalized processes: exponential inequalities, moment bounds and iterated logarithm laws. Annals of probability, pages 1902–1933.

Derman, E., Mankowitz, D., Mann, T., and Mannor, S. (2020). A bayesian approach to robust reinforcement learning. In Uncertainty in Artificial Intelligence, pages 648–658. PMLR.

Donti, P., Amos, B., and Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. In Advances in Neural Information Processing Systems, pages 5484–5494.

Dunlop, M. D. (2019). Ontology-driven, adaptive, medical questionnaires for patients with mild learning disabilities. In Artificial Intelligence XXXVI: 39th SGAI International Conference on Artificial Intelligence, AI 2019, Cambridge, UK, December 17–19, 2019, Proceedings, page 107. Springer.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research, 16(1):1437–1480.

Garivier, A. and Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. In International Conference on Algorithmic Learning Theory, pages 174–188. Springer.

Goldenshluger, A. and Zeevi, A. (2013). A linear response bandit problem. Stochastic Systems, 3(1):230–261.

Gottesman, O., Johansson, F., Meier, J., Dent, J., Lee, D., Srinivasan, S., Zhang, L., Ding, Y., Wihl, D., Peng, X., et al. (2018). Evaluating reinforcement learning algorithms in observational health settings. arXiv preprint arXiv:1805.12298.

Günlük, O., Kalagnanam, J., Menickelly, M., and Scheinberg, K. (2018). Optimal decision trees for categorical data via integer programming. arXiv preprint arXiv:1612.03225.

Haijema, R., van der Wal, J., and van Dijk, N. M. (2007). Blood platelet production: Optimization by dynamic programming and simulation. Computers & Operations Research, 34(3):760–779.

Harchaoui, Z., Moulines, E., and Bach, F. R. (2009). Kernel change-point analysis. In Advances in neural information processing systems, pages 609–616.

Harrison Jr, D. and Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.

Heaven, W. D. (2020 (accessed October 15th, 2020)). Our weird behavior during the pandemic is messing with AI models.

Huang, C., Zhai, S., Talbott, W., Bautista, M. A., Sun, S.-Y., Guestrin, C., and Susskind, J. (2019). Addressing the loss-metric mismatch with adaptive loss alignment. arXiv preprint arXiv:1905.05895.

Javad, M. O. M., Agboola, S. O., Jethwani, K., Zeid, A., and Kamarthi, S. (2019). A reinforcement learning–based method for management of type 1 diabetes: Exploratory study. JMIR diabetes, 4(3):e12905.

Jiang, D. R. and Powell, W. B. (2015). Optimal hour-ahead bidding in the real-time electricity market with battery storage using approximate dynamic programming. INFORMS Journal on Computing, 27(3):525–543.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial intelligence, 101(1-2):99–134.

Kannan, S., Morgenstern, J., Roth, A., Waggoner, B., and Wu, Z. S. (2018). A smoothed analysis of the greedy algorithm for the linear contextual bandit problem. arXiv preprint arXiv:1801.03423.

Kovatchev, B. P., Breton, M., Dalla Man, C., and Cobelli, C. (2009). In silico preclinical trials: a proof of concept in closed-loop control of type 1 diabetes.

Kovatchev, B. P., Cox, D. J., Gonder-Frederick, L. A., and Clarke, W. (1997). Symmetrization of the blood glucose measurement scale and its applications. Diabetes Care, 20(11):1655–1658.

Kovatchev, B. P., Straume, M., Cox, D. J., and Farhy, L. S. (2000). Risk analysis of blood glucose data: a quantitative approach to optimizing the control of insulin dependent diabetes. Computational and Mathematical Methods in Medicine, 3(1):1–10.

Köhler, S. and al (2016). The Human Phenotype Ontology in 2017. Nucleic Acids Research, 45.

Lai, T. L. (1995). Sequential changepoint detection in quality control and dynamical systems. Journal of the Royal Statistical Society: Series B (Methodological), 57(4):613–644.

Lai, T. L. and Wei, C. Z. (1982). Least squares estimates in stochastic regression models with applications to identification and control of dynamic systems. The Annals of Statistics, 10(1):154–166.

Lattimore, T. and Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.

Lebarbier, É. (2005). Detecting multiple change-points in the mean of gaussian process by model selection. Signal processing, 85(4):717–736.

Lecarpentier, E. and Rachelson, E. (2020). Non-stationary markov decision processes, a worst-case approach using model-based reinforcement learning, extended version.

Li, Y. (2018). Deep reinforcement learning. CoRR, abs/1810.06339.

Lu, C., Schölkopf, B., and Hernández-Lobato, J. M. (2018). Deconfounding reinforcement learning in observational settings. arXiv preprint arXiv:1812.10576.

Magelssen, M., Supphellen, M., Nortvedt, P., and Materstvedt, L. J. (2016). Attitudes towards assisted dying are influenced by question wording and order: a survey experiment. BMC medical ethics, 17(1):24.

McNellis, J. (2020 (accessed October 15th, 2020)). Unprecedented times + machine learning = poor experiences?

Meisel, S. and Powell, W. B. (2017). Dynamic decision making in energy systems with storage and renewable energy sources. In Bertsch, V., Fichtner, W., Heuveline, V., and Leibfried, T., editors, Advances in Energy System Optimization, pages 87–101, Cham. Springer International Publishing.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Mwamikazi, E., Fournier-Viger, P., Moghrabi, C., Barhoumi, A., and Baudouin, R. (2014). An adaptive questionnaire for automatic identification of learning styles. In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pages 399–409. Springer.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In Icml, volume 1, page 2.

Ngo, P. D., Wei, S., Holubová, A., Muzik, J., and Godtliebsen, F. (2018). Control of blood glucose for type-1 diabetes by using reinforcement learning with feedforward algorithm. Computational and mathematical methods in medicine, 2018.

Nguengang, W. S. and al (2019). Estimating cumulative point prevalence of rare diseases: analysis of the orphanet database. Eur J Hum Genet.

Nokelainen, P., Niemivirta, M., Kurhila, J., Miettinen, M., Silander, T., and Tirri, H. (2001). Implementation of an adaptive questionnaire. In Proceedings of the ED-MEDIA Conference, pages 1412–1413.

Nunes, C., De Craene, M., Langet, H., Camara, O., and Jonsson, A. (2020). Learning decision trees through monte carlo tree search: An empirical evaluation. WIREs Data Mining and Knowledge Discovery, 10(3):e1348.

Plazas, M. A., Conejo, A. J., and Prieto, F. J. (2005). Multimarket optimal bidding for a power producer. IEEE Transactions on Power Systems, 20(4):2041–2050.

Prodel, M. (2017). Process discovery, analysis and simulation of clinical pathways using health-care data. Theses, Université de Lyon.

Provost, F., Melville, P., and Saar-Tsechansky, M. (2007). Data acquisition and cost-effective predictive modeling: targeting offers for electronic commerce. In Proceedings of the ninth international conference on Electronic commerce, pages 389–398.

Puterman, M. (2014). Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics. Wiley.

Rusmevichientong, P. and Tsitsiklis (2010). Linearly parameterized bandits. arXiv preprint arXiv:0812.3465.

Sarma, P., Chen, W. H., Durlofsky, L. J., Aziz, K., et al. (2006). Production optimization with adjoint models under nonlinear control-state path inequality constraints. In Intelligent Energy Conference and Exhibition. Society of Petroleum Engineers.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 362(6419):1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. nature, 550(7676):354–359.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). Conditional variable importance for random forests. BMC bioinformatics, 9(1):307.

Sun, Q., Jankovic, M. V., Budzinski, J., Moore, B., Diem, P., Stettler, C., and Mougiakakou, S. G. (2018). A dual mode adaptive basal-bolus advisor based on reinforcement learning. IEEE journal of biomedical and health informatics, 23(6):2633–2641.

Sun, Q., Jankovic, M. V., and Mougiakakou, S. G. (2019). Reinforcement learning-based adaptive insulin advisor for individuals with type 1 diabetes patients under multiple daily injections therapy. In 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pages 3609–3612. IEEE.

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

The Epsilon Group (2017). Services Diabetes Simulation. https://tegvirginia.com/services/diabetes-simulation/. [Online; accessed 05-December-2019].

Tomašev, N., Paquet, U., Hassabis, D., and Kramnik, V. (2020). Assessing game balance with alphazero: Exploring alternative rule sets in chess.

Tremblay, M., Colbaugh, R., Glass, K., and Rudolf, C. (2018). Robust ensemble learning to identify rare disease patients from electronic health records.

Tropp, J. A. (2011). User-friendly tail bounds for matrix martingales. Technical report, California Institute of Technology, Pasadena, CA.

Truong, C., Oudre, L., and Vayatis, N. (2020). Selective review of offline change point detection methods. Signal Processing, 167:107299.

Tsechpenakis, G., Metaxas, D. N., Neidle, C., and Hadjiliadis, O. (2006). Robust online change-point detection in video sequences. In Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop, pages 155–161.

Van Moffaert, K. and Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. The Journal of Machine Learning Research, 15(1):3483–3512.

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., and Schaus, P. (2019). Learning optimal decision trees using constraint programming. In BNAIC/BENELEARN.

Wang, C., Li, G., Reynolds, A. C., et al. (2009). Production optimization in closed-loop reservoir management. SPE journal, 14(03):506–523.

Weckert, S. (2020 (accessed October 15th, 2020)). Google Maps Hacks.

Xie, J. (2018). Simglucose v0.2.1. https://github.com/jxx123/simglucose. [Online; accessed 05-December-2019].

Yang, R., Sun, X., and Narasimhan, K. (2019). A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems 32, pages 14636–14647. Curran Associates, Inc.

Zhang, J., Zhang, X., Sun, K., Yang, X., Dai, C., and Guo, Y. (2019). Unsupervised annotation of phenotypic abnormalities via semantic latent representations on electronic health records.

INSTITUT POLYTECHNIQUE DE PARIS
École doctorale de mathématiques Hadamard (EDMH)

Titre : Quelques contributions à des problèmes de décisions

Mots clés : Processus Décisionnel de Markov, Fonction de perte, Apprentissage par renforcement, Diabète, Plan de production, Questionnaire Adaptatif

Résumé : Cette thèse, motivée par des applications des secteurs de l'industrie et de la santé, est un recueil d'études sur différents problèmes de décision.

Dans la première partie, nous nous concentrons sur des problèmes de prise de décision en une seule étape, où un modèle prédictif est utilisé en amont de la prise de décision, et où un retour d'information explicite est reçu. Nous proposons de laisser à l'utilisateur final la tâche de définir la fonction de perte associée au modèle prédictif, de façon à encoder le coût réel de l'utilisation d'une prévision pour prendre une décision. Comme approche algorithmique, nous considérons les arbres de décisions, optimisés avec la fonction de perte ajustée, et des méthodes d'approximation de fonction, liée à l'apprentissage des Q-valeurs dans l'apprentissage par renforcement, dans le cas où seule la récompense immédiate est d'intérêt. Trois applications sont étudiées : le calibrage d'un système d'alarme pour lutter contre l'errance médicale ; le problème de la nomination sur un marché de l'électricité, du point de vue d'un fournisseur d'énergies renouvelables ; l'optimisation de la production dans l'incertitude de la demande des clients.

Dans la deuxième partie, nous nous intéressons essentiellement à deux problèmes spécifiques de prise de décision séquentielle, que nous abordons à l'aide d'un cadre de processus décisionnel de Markov et d'algorithmes d'apprentissage par renforcement. Dans la première application, nous essayons d'optimiser les moment de repas et la gestion de l'insuline pour les personnes souffrant de diabète de type I et qui comptent sur les auto-injections. Pour ce faire, nous nous appuyons sur un simulateur de patient, lequel est basé sur la connaissance médicale de l'interaction entre glucose et insuline et sur des paramètres physiologiques propres aux patients. Dans la seconde application, nous essayons de construire un questionnaire prédictif adaptatif pour des interactions lisses avec les utilisateurs. Pour des données binaires, le questionnaire ressemble à un arbre de décision, optimisé de façon bottom-up. Pour des données non-binaires, ce nouveau questionnaire ne redemande des questions déjà posées, se souvient des valeurs observées précédemment, et les exploite pleinement une fois arrivé dans un noeud terminal, où une fonction de prédiction spécifique est disponible.

Dans notre dernière partie, nous nous intéressons à trois processus de décision qui, par construction, n'exigent pas que l'agent explore l'environnement. Par exemple, nous considérons un système dont la dynamique est suffisamment stochastique pour que, quelle que soit notre action, nous explorions l'espace d'état, tout en ayant une certaine influence par nos actions. Nous considérons également un système où certaines actions sont indisponibles aléatoirement en fonction des epochs. Outre les résultats théoriques trouvés, cette partie met l'accent sur l'importance de concentrer l'exploration là où elle est nécessaire.

Title : Some contributions to decision-making problems

Keywords : Markov Decision Process, Loss function, Reinforcement Learning, Diabetes, Production Plans, Adaptive Questionnaire

Abstract : This thesis, motivated by applications in the industrial and health sectors, is a collection of studies on different decision problems.

In the first part, we focus on single-step decision-making problems where a predictive model is used upstream of the decision-making, and where explicit feedback is received. We propose to leave the task of defining the loss function associated with the predictive model to the end-user, in order to encode the real cost of using a forecast to make a decision. As an algorithmic approach, we consider decision trees, optimized with the adjusted loss function, and function approximation methods, similar to Q-value function approximation in reinforcement learning, in case where only the immediate reward is of interest. Three applications are studied : the calibration of an alarm system to fight against medical wandering ; the problem of nomination in an electricity market, from the point of view of a renewable energy supplier ; the optimization of production in the uncertainty of customer demand.

In the second part, we focus on two specific problems of sequential decision-making, which we address using a Markov decision-making framework and reinforcement learning algorithms. In the first application, we try to optimize meal timing and insulin management for people with type I diabetes who rely on self-injections. To do so, we rely on a patient simulator, which is based on medical knowledge of the interaction between glucose and insulin and on physiological parameters specific to the patients. In the second application, we try to build an adaptive predictive questionnaire for smooth interactions with users. For binary data, the questionnaire looks like a decision tree, optimized in a bottom-up way. For non-binary data, this new questionnaire only asks questions that have already been asked, remembers previously observed values, and exploits them fully once they arrive in a terminal node, where a specific prediction function is available.

In our final section, we look at three decision processes that, by construction, do not require the agent to explore the environment. For example, we consider a system whose dynamics are sufficiently stochastic that, whatever our action, we explore the state space, while having some influence through our actions. We also consider a system where some actions are randomly unavailable depending on the epochs. In addition to the theoretical results found, this part emphasizes the importance of focusing the exploration where it is needed.

Institut Polytechnique de Paris
91120 Palaiseau, France