Reinforcement Learning and Artificial Intelligence in the context of revenue management

Freie wissenschaftliche Arbeit
zur Erlangung des akademischen Grades
"Master of Science"
Studiengang: Finanz- und Informationsmanagement

an der
Wirtschaftswissenschaftlichen Fakultät
der Universität Augsburg

– Lehrstuhl für Analytics & Optimization –

Eingereicht bei:	Prof. Dr. Robert Klein
Betreuer:	Dr. Sebastian Koch
Vorgelegt von:	Stefan Glogger
Adresse:	Riedstr. 21
	86647 Buttenwiesen
Matrikel-Nr.:	1471113
E-Mail:	stefan.glogger@tum.de
Ort, Datum:	Augsburg, 30. September 2019

Acknowledgement

First of all, I want to thank my supervisor Prof. Robert Klein for giving me the opportunity for this thesis, the complaisant general conditions and his helpful advice during the working process. Prof. Klein has given me the possibility to explore the field of revenue management and to apply many of the distinct areas taught in my master degree program “Finance and Information Management”.

Moreover, I want to thank my advisor Sebastian Koch for supporting me throughout the process of this thesis. Especially the focused attitude during our meetings and the overall efficient communication were very beneficial. He also gave me the freedom to incorporate my own ideas in the thesis and to work independently.

Lastly, and most importantly, I am grateful for my family and my close friends for their love, support, time and steady encouragement.

Contents

List of Figures

List of Tables

List of symbols

Symbols of Chapter 2

Symbol	Characteristic	Meaning
i, j, h, t, l, p	general small letter	index
n, m	specific small letters	maximum index
$[n]$	set of natural numbers	$\{1, \dots, n\}$
T, L	general capital letter	maximum index
S	specific capital letter	tuple of elements
x_1	small with index	single number
$\boldsymbol{x}$	small and bold	vector
$(x_1, \dots, x_n)$	parenthesis around single numbers	row vector
$(\dots)^\intercal$	$^\intercal$ on upper side	transpose

Table 1: Concepts of notation

Symbol	Domain	Meaning
n	$\mathbb{N}$	total number of products
j	$[n]$	index for the n products
$\boldsymbol{r}$	$\mathbb{R}^n$	revenue vector for the n products
m	$\mathbb{N}$	total number of resources
h	$[m]$	index for the m resources
$\boldsymbol{a}_j$	$\{0,1\}^m$	vector of needed resources for product j
$A = (a_{hj})$	$\{0,1\}^{m \times n}$	matrix representing all needed resources for all products
L	$\mathbb{N}$	total number of customer segments
l	$[L]$	index for the L customer segments
λ_l	$\mathbb{R}_0^+$	arrival probability for customer of segment l
T	$\mathbb{N}$	total number of time periods
t	$[T]$	index for time periods
c_h^0	$\mathbb{N}$	initial capacity of resource h
$\boldsymbol{c}^0$	$\mathbb{N}^m$	vector of initial capacities for all resources
c_h^t	$\{0, \dots, c_h^0\}$	capacity of resource h available at the end of time period t
$\boldsymbol{c}^t$	$\mathbb{N}^m$	vector of capacities for all resources available at the end of time period t
x_j^t	$\{0,1\}$	indicates if during time period t product j is offered (1) or not (0)
$\boldsymbol{x}^t$	$\{0,1\}^m$	offer tuple of products during time period t
$p_{\boldsymbol{x}^t}(j)$	$[0,1]$	probability of product j being purchased when tuple $\boldsymbol{x}^t$ is offered
$V(t, \boldsymbol{c})$	$\mathbb{R}_0^+$	value function evaluated at state $(t, \boldsymbol{c})$
$\hat{\boldsymbol{v}}^t$	$\{0,1\}^m$	optimal offer tuple of products to be offered during time period t
u_{lj}	$\mathbb{R}_0^+$	utility that a customer of segment l assigns to product j
u_{l0}	$\mathbb{R}_0^+$	utility that a customer of segment l assigns to purchasing nothing at all
$R(\boldsymbol{x})$	$\mathbb{R}_0^+$	expected revenue given offer tuple $\boldsymbol{x}$
$Q_h(\boldsymbol{x})$	$\mathbb{R}_0^+$	expected capacity of resource h being used given offer tuple $\boldsymbol{x}$
$\boldsymbol{Q}(\boldsymbol{x})$	$\mathbb{R}_0^{+m}$	expected capacities being used given offer tuple $\boldsymbol{x}$
$t(\boldsymbol{x})$	$\mathbb{R}_0^+$	total time for which offer tuple $\boldsymbol{x}$ is offered
N	$\{0,1\}^n$	set of all possible offer tuples
$V^{CDLP}(T, \boldsymbol{c})$		choice-based linear program
$VD^{CDLP}(T, \boldsymbol{c})$		dual of choice-based linear program
$\boldsymbol{\pi}$	$\mathbb{R}^m$	dual variables corresponding to resource constraints
σ	$\mathbb{R}$	dual variable corresponding to time constraint
$V^{CDLP-R}(T, \boldsymbol{c})$		reduced choice-based linear program
$VD^{CDLP-R}(T, \boldsymbol{c})$		reduced dual of choice-based linear program
$rc(\boldsymbol{x})$		reduced costs of offer tuple $\boldsymbol{x}$

Table 2: Chapter 2 - overview of parameters.

Symbols of Chapter 3

Symbol	Domain	Meaning
${}^{\pi}\boldsymbol{x}^t$	$\{0,1\}^n$	optimal offer tuple to be offered during time period t
θ_t	$\mathbb{R}$	optimization variable (offset) for time period t
Θ	$\mathbb{R}^{\intercal}$	optimization variable comprising all θ_t
$\boldsymbol{\pi}_t$	$\mathbb{R}^m$	optimization variable (bid price for each resource) for time period t
$\boldsymbol{\Pi}$	$\mathbb{R}^{m\times T}$	optimization variable (bid price for each resource) comprising all $\boldsymbol{\pi}_t$
$\hat{V}_t^i$	$\mathbb{R}$	sample revenue to go for sample i starting at and including period t
$\hat{V}$	$\mathbb{R}^{T\times I}$	all sample revenues to go comprising all $\hat{V}_t^i$
$\hat{\boldsymbol{C}}_{t-1}^i$	$\mathbb{R}^m$	available capacities for resources for sample i at end of period $t-1$
$\hat{\boldsymbol{C}}$	$\mathbb{R}^{m\times T\times I}$	all available capacities comprising all $\hat{\boldsymbol{C}}_{t-1}^i$
r_t	$\mathbb{R}$	sample revenue generated during time period t
$\boldsymbol{c}$	$\mathbb{N}^m$	available capacities for each resource at current time period
$\boldsymbol{c}^0$	$\mathbb{N}^m$	starting capacities
$\boldsymbol{x}$	$\{0,1\}^n$	offerset at current time period
ϵ_t	$[0,1]$	epsilon used at time period t

Table 3: Chapter 2 - overview of parameters.

Symbols of Chapter 4

Symbol	Domain	Meaning
D	$\mathbb{N}$	dimension of input space
N	$\mathbb{N}$	number of data points
$\boldsymbol{x}$	$\mathbb{R}^D$	feature data
y	$\mathbb{R}$	value
$\boldsymbol{X}$	$\mathbb{R}^{N \times D}$	design matrix comprising all training data points
$\boldsymbol{y}$	$\mathbb{R}^N$	vector of training labels
$\boldsymbol{w}$	$\mathbb{R}^D$	weight vector
$f_{\boldsymbol{w}}(\boldsymbol{x})$		linear function
$E(\cdot)$		error function
$\boldsymbol{w}^*$	$\mathbb{R}^D$	optimal weight vector
∇		Nabla-operator to indicate gradient
$i_n^{(i)}$	$\mathbb{R}$	input variable for layer i, node n
$o_n^{(i)}$	$\mathbb{R}$	output variable for layer i, node n
$w_{jk}^{(i)}$	$\mathbb{R}$	weight for layer $i \geq 2$, to be multiplied with $o_k^{(i-1)}$
$f(\cdot)$		activation function
$\boldsymbol{W}$		tensor comprising all weights of the neural network
$n_{\boldsymbol{W}}(\boldsymbol{x})$		applying a neural network with weights $\boldsymbol{W}$ to variable $\boldsymbol{x}$

Table 4: Chapter 4 - overview of parameters.

Symbols of Chapter 5

Symbol	Domain	Meaning
K	$\mathbb{N}$	number of sample paths
$\boldsymbol{v}^A$	$\mathbb{R}^K$	vector of values generated by policy A
V^A		random variable associated with policy A
F^A		distribution of random variable associated with policy A
H_0		null hypothesis
H_1		alternative hypothesis
α		significance level
$N(0,1)$		standard normal distribution
B_{np}		binomial distribution with sample size n and parameter p

Table 5: Chapter 5 - overview of parameters.

Chapter 1

Introduction

This document presents my Master's Thesis for the M. Sc. Programme *Finance and Information Management*, in which I explain, implement and compare the performance of various techniques and algorithms in revenue management. This first chapter gives a short overview of the history of revenue management, introduces the problem to be discussed in the following chapters and puts the thesis in context of different fields of revenue management. It furthermore gives a short overview of the chapters to come and briefly explains how I see this thesis fitting in the context of my study programmes.

The concept of revenue management originated as a result of the U. S. Airline Deregulation Act of 1978. Prior to this, prices had been strictly controlled and standardized by the U. S. Civil Aviation Board (CAB). Afterwards, airlines were free to choose prices, change schedules and alter offered services at their will, without CAB approval. This deregulation resulted in an immediate appearance of low-cost/no-frill airlines. By operating on lower labour costs and providing less service on board, profitable fares up to 70% lower than the ones of major carriers could be offered and market pressure on classical airlines increased. Consequently, all airline companies faced the problem of which price to offer at any given circumstances.

The traditional assumption of independent demand didn't work. It expects demand to be independent of the current market conditions such as prices offered by competitors, (day-)time of departure, frequency of departures, brand image or others, to be found in e. g. Talluri and Ryzin (2005). Furthermore, low fare demand is generally expected to precede high fare demand, as business travellers prefer the flexibility to possibly change their schedule. Thus, the problem reduced to: How much capacity should be reserved for the high fare demand, that appears later? But as the purchase of a ticket clearly depends on the market situation (e. g. an exceptional Formula 1 race at one specific weekend) or

the continuing expansion of low-cost airlines offering undifferentiated fare structures, this simple assumption of independence falls short in practice as also stated in Bront et al. (2009).

As a result, academia shifted focus to more sophisticated models building on customer choice behaviour. Demand is assumed to depend on the currently offered products, which might change depending on the circumstances. This assumption now also accounts for buy-up (buying a higher fare when lower fares are closed) and buy-down (switching to a lower fare when discounts are available). The assumption of customer choice is then incorporated in dynamic pricing models, e. g. as studied by Gallego and van Ryzin (1997), Bitran et al. (1998) or Feng and Gallego (2000).

The underlying problem is referred to as *capacity control under customer choice behaviour* and can be summarized as: Which products should be offered at any given circumstance? In the airline setting, circumstance is determined mainly by time until departure and remaining capacity. As many airlines operate so called *hub-and-spoke networks*, allowing them to offer service in many more markets than with point-to point connection, the problem becomes more challenging as stated e. g. in Talluri and Ryzin (2005). This network structure can be incorporated by modelling each point-to-point connection (single leg) by one separate resource and each seat as one unit of capacity. Furthermore, different customers (e. g. business vs. leisure traveller) can be distinguished by modelling them as different customer segments. Thus, the central problem in capacity control can be stated as:

In the setting of a network consisting of several flights (edges) connecting certain cities (vertices), when given a particular seat capacity (weight of edge) and a certain time until departure, which combination of products shall be offered to customers?

Note that revenue management is relevant in many fields, even though we introduced the problem purely in the airline setting. The introduction focussed on this industry as revenue management is closely connected to this particular industry and it increases total profitability by 4 to 5% of revenues in comparison with not using any revenue management system as stated in Talluri and Ryzin (2005)[1]. However, revenue management appears in any decision situation related to demand management. Talluri and Ryzin (2005) structured those decisions in three basic categories:

[1] We want to point out that Southwest Airlines is often mentioned as a counterexample. But even though its pricing structure is very simple, revenue management systems are still used as pointed out by Talluri and Ryzin (2005).

- Structural decisions: How the selling process is organized (e. g. negotiations among individuals, auctions with restricted access, publicly posted prices) or how additional terms are structured (e. g. volume discounts or refund options).
- Price decisions: How to set individual offer prices, reserve prices (in auctions) and posted prices or how to price distinct products over time.
- Quantity decisions: Whether an offer to buy should be accepted or rejected or when to withhold a product from the market in order to sell it at a later point in time.

For this thesis, structural and price decisions have been made already: Prices are fixed in advance and posted publicly. The quantity decision remains and we can now formulate the goal of this thesis. We want to explore different methods of "solving" the capacity control problem under choice behaviour in a single-leg and multi-leg setting by implementing the algorithms, letting them cope with exemplary test scenarios and comparing their performances. Furthermore, new methods such as the usage of Neural Networks shall be discussed and evaluated.

The thesis is structured as follows. After having given a short introduction, Chapter 2 describes the problem formally and introduces standard solution methods. Chapter 3 presents a modern solution method, namely Approximate Dynamic Programming. Chapter 4 presents a newly developed algorithm based on Neural Networks. Chapter 5 gives theory on how to compare different methods based on statistics and compares a total of eight different algorithms via three statistical procedures. Chapter 6 presents further discoveries made while working on this thesis. Finally, Chapter 7 summarizes the thesis and presents an outlook for future research. More mathematical background is given in Appendix A. The code of the implementation is outlined in Appendix C and can be found on my GitHub-Repository[2].

Briefly, I want to elaborate on how I see this thesis fitting into my study programmes and my personal goals going along with it. This thesis shall combine the programme "Finance and Information Management", with its interdisciplinary components of Business, Mathematics and Informatics, together with my personal interests and professional working attitude. The optimization component is covered by modelling the situation, creating an optimization problem, implementing this in software, visualizing the results and applying duality theory. The business component can be found in concepts such as utility

[2] https://github.com/gloggerS/RL-and-AI-in-Revenue-Management.git

functions, complementary assets and substitutes. Basic probability theory is incorporated with probability spaces, Kolmogorov-axioms, random variables and markov chains. Concepts of statistics are included as statistical tests are created, implemented and results presented. The informatics component is covered by preparing code in the modern programming language Python[3] and following modern software standards, as introduced by The Zen of Python[4], the style guide PEP 8[5] and reproducibility. The mathematical component can be found throughout the thesis by precise notation, e. g. by reformulating of some concepts found in papers to make them more precise, and by elaborating in a more mathematical way in footnotes or the appendix. The thesis should be well written, properly set in LaTeX and showing my further enhanced typographic and layout skills, also in TikZ. Overall, this thesis presents a concise overview of techniques currently applied in revenue management and can be used as a starting point for future research in this field.

[3] Python is already heavily used in the Data Science community and its popularity is still increasing. Thus, many high quality libraries are available.
[4] https://www.python.org/dev/peps/pep-0020/
[5] https://www.python.org/dev/peps/pep-0008/

Chapter 2

Problem and standard approaches to solution

This chapter introduces the classical revenue management problem of capacity control under customer choice behaviour as can be found e. g. in Koch (2017) or in Strauss et al. (2018). The problem consists of deciding which products to offer at any given point in time and Section 2.1 describes it in general as well as formulates it as a dynamic program. Two standard methods of solving the problem are laid out in the following. The exact solution is described in Section 2.2 and choice based linear programming in Section 2.3. Duality theory and other techniques are applied to speed up the calculation even more.

2.1 The problem

This section describes the capacity control in general in Section 2.1.1. It furthermore presents a concrete example in the airline setting in Section 2.1.2 and introduces the formulation of the resulting choice based capacity control model as a dynamic program in Section 2.1.3.

2.1.1 General description

We cover a setting that is most regularly seen in every-day life: A company offers a bundle of products to heterogeneous customers arriving over time. Let us now introduce some terminology. *Products* often correspond to services that have to be delivered at a given point in time. The period of time from now until delivery is referred to as *selling horizon* (or booking horizon), is considered finite and is split into a finite set of consecutive *time*

points. The set of available products is also assumed to be finite and without loss of generality (w. l. o. g.) constant over time[1]. Each individual product is characterized by a given *price* and usage of certain, finitely many resources. These *resources* might be shared among different products.

The goal of most companies is to increase profits, which is equivalent to maximizing revenue. As stated, the price of individual products (leading to revenue) is assumed to be fixed, which is a realistic assumption for short terms considering potentially large costs coming with a change of price (e. g. selling products via catalogue). Furthermore, capacities are fixed in the short term (costly to increase production as e. g. more employees have to be hired) and are considered perishable. Often, capacity comes with high fixed costs and low marginal costs resulting from selling an additional product at any price (e. g. costs for operating one flight with 97 customers are usually the same as for operating the same flight with 98 customers). Such a profit and cost structure justifies the usage of revenue as a proxy for profit as also stated in Strauss et al. (2018).

Switching to the customer side, the consideration of a heterogeneous customer group allows for modelling most real life scenarios. Not every customer is the same, but it is certainly reasonable to combine individuals into groups sharing similar characteristics. Some characteristics will be more prevalent in the overall population (more precisely, the group of people that represents all persons interested in the companies' products). Different customers have varying needs, thus different preferences and different willingness-to-pay, leading ultimately to different purchases depending on the currently offered set of products. Thus, the company can influence the sales process by altering the offered set of products over the booking horizon, which is exactly the task of capacity control under customer choice behaviour.

2.1.2 Description in airline setting

The concepts introduced above are now applied to a concrete airline setting, which will later be used to present the solution techniques. The airline company AirCo operates a flight network connecting cities A, B and C as depicted in Figure 2.1a. The booking horizon consists of 20 time steps (e. g. today is Monday, booking is possible four times every day until and including Friday and flights depart on Saturday). The products to be

[1] If the set of available products change over time, let S_t denote the set of available products at time t and let $\mathcal{T}$ represent the finite set of time points. Then, $S \coloneqq \cup_{t \in \mathcal{T}} S_t$ is again finite (finite union of finite sets) and can be taken as the constant set of available products at each time point.

offered are summarized in Figure 2.1b. Note that two products are available on Leg 1, i. e. the expensive product 1 and the less expensive product 2 (e. g. due to separate business and economy classes). A sell of either product 1 or product 2 will result in the reduction of available capacity on Leg 1 by one unit. Furthermore, product 6 presents an option to get from city A to city C (via city B) and thus a sale of product 6 results in a reduction of capacities on both Leg 1 and Leg 2 by one unit. Also product 4 is connecting city A and city C (on the direct flight) but is more expensive then product 6 (e. g. the company assumes the shorter travel time is of value to some travellers).

Furthermore, capacities are presented in Figure 2.1c, so there are eight available seats on Leg 1, but just four seats are available on Leg 2. Customers are characterized as stated in Figure 2.1d, so most customers belong to segment 1 (higher value of λ_l representing larger size of customer segment l), want to travel from city A to city B and prefer the budget option (as the value of preference for the latter product is much higher and a higher value corresponds to a higher preference). Customers belonging to segment 2 also want to travel from A to B, but are price insensitive, as they value the higher comfort of the more expensive product 1 just slightly more than the cheaper product 2. Customers that have to travel from A to C are represented by customer segment 3 as can be seen by the low preference for no-purchase.

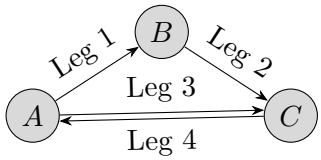
2.1.3 Formulation as dynamic program

Extensive literature deals with revenue management and makes use of formal notation suitable for the analysis of capacity control models. Good overviews can be found in Klein and Steinhardt (2008) or in Phillips (2011). In this thesis, we introduce notation for a capacity control model first under a general discrete choice model of demand and then directly using a multinomial logit model of demand. Notation is closely linked to Koch (2017) and the standard representation is used for a better orientation with the main ideas outlined in Table 1 and a concrete overview on parameters is given in Table 2.

Formal notation

First, we present the situation with its inflexible components. Consider a firm that produces n distinguishable products indexed by $j = 1, \dots, n$. We introduce the notation $[n] := \{1, \dots, n\}$. Product j comes with a certain price leading to a revenue of r_j when the product is sold. The revenues are bundled by the vector $\boldsymbol{r} = (r_1, \dots, r_n)^\intercal$. A total of

(a) Airline network.



(b) Products.

Product	Origin-destination	Fare
1	$A \to B$	5
2	$A \to B$	3
3	$B \to C$	5
4	$A \to C$	12
5	$C \to A$	10
6	$A \to B \to C$	8

(c) Resources.

Leg	Capacity
1	8
2	4
3	4
4	8

(d) Customers.

Seg.	λ_l	Consideration tuple[a]	Preference vector	No purchase preference	Description
1	0.3	$(1, 2)$	$(4, 8)$	2	Price sensitive, (A→B)
2	0.2	$(1, 2)$	$(6, 5)$	2	Price insensitive, (A→B)
3	0.2	$(4, 6)$	$(8, 4)$	1	Price sensitive, fast (A→C)
4	0.1	(5)	(2)	2	Opportunity traveller, (C→A)

[a]Note that in contrast to Bront et al. (2009), we use the mathematically correct terminology as *tuple* does have an inherent order, while *set* does not (and thus combining a set of products with a set of preferences is mathematically not correct).

Figure 2.1: Working example to illustrate problem with $T = 20$ time periods.

m distinct resources are used for production, which are indexed by $h \in [m]$. In order to produce one unit of product j, certain resources are needed and the consumption of resources is captured by the vector $\boldsymbol{a}_j = (a_{1j}, \ldots, a_{mj})^\intercal$, with $a_{hj} = 1$ if resource h is needed for production of product j and $a_{hj} = 0$ otherwise. The resource consumption can be aggregated in the incidence matrix $A = (a_{hj})_{h\in[m],j\in[n]} \in \{0, 1\}^{m\times n}$. In addition, the firm already identified the group of potential customers and segmented this group into a total of L customer segments, which are indexed by $l \in [L]$. A customer of segment l arrives with probability λ_l and this parameter can be adjusted such that it matches the reality (e. g. there are twice as many customers belonging to segment a as there are of segment b, so we choose $\lambda_a = 2\lambda_b$) and the model assumptions (only one customer arrives during each period).[2]

The finite time period from when booking is possible for the first time until when it is possible for the last time is discretized into a total of T sufficiently small time periods, such that in each individual time period at most one customer arrives. Time periods are

[2]Note that this essentially means that the arrival of customers of an individual segment is following a Poisson distribution and more details on the mathematics involved can be found in Appendix A.2.

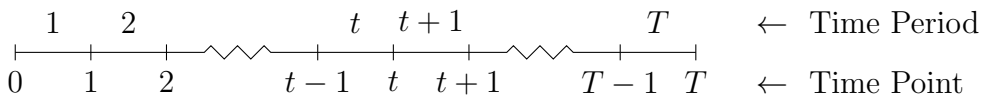


Figure 2.2: Visualization of booking horizon with its time periods vs. its time points.

indexed by $t \in [T]$. To be very precise (particularly relevant for correct implementation), booking itself is just possible at the beginning of a time period, i. e. for time period t, booking is possible at time point $t-1$. Furthermore, one customer purchases at most one product.

Secondly, we look at the flexible components of the model, i. e. those components that might change over time. Initially, i. e. at the start of the first time period (time point 0), the available capacity of resource h is given by c_h^0 and thus all available capacities are described by the vector $\boldsymbol{c}^0 = (c_1^0, \dots, c_m^0)^\intercal$.

Let us now explore what happens if product j is purchased during time period t, i. e. at time point $t-1$. The old capacity vector is given by $\boldsymbol{c}^{t-1}$. As product j claims capacities according to $\boldsymbol{a}_j$, the capacity vector reduces accordingly to $\boldsymbol{c}^t = \boldsymbol{c}^{t-1} - \boldsymbol{a}_j$. Time moves forward, such that the last selling might occur during time period T, i. e. at time point $T-1$.

The distinguishment of time period and time point is also visualized in Figure 2.2. Note that time period t can be seen as the half open interval between consecutive time points: $(t-1, t]$.

In summary, the firm aims at increasing the value of the products sold and has flexibility in the sets offered. Thus, the decision variables for each time period $t \in [T]$ are given by $\boldsymbol{x}^t = (x_1^t, \dots, x_n^t)^\intercal$ with $x_j^t = 1$ if product j is offered during time period t (i. e. chosen at time point $t-1$) and $x_j^t = 0$ otherwise. Thus, for each time period t and the prevalent capacity $\boldsymbol{c}$, the offer set $\boldsymbol{x}$ has to be determined. Note that in machine learning terms, this mapping is defined as a policy $\boldsymbol{\psi}(\boldsymbol{c}, t)$.

We can put the goal mentioned above in formulas after introducing the random variable X^t as the product which is purchased at time point t, with the notation $X^t = \{0\}$ if no-purchase is made at time point t. So we use $\Omega := \{0, 1, \dots n\}$ as the underlying set (no-purchase plus all purchasable products) and can use $\mathcal{F} = \mathrm{P}(\Omega)$ as underlying sigma algebra, with $\mathrm{P}(\cdot)$ describing the power set[3]. Furthermore, we introduce a probability measure $p_{\boldsymbol{x}^t}(\cdot)$ and use the shorthand $p_{\boldsymbol{x}^t}(j) = p_{\boldsymbol{x}^t}(\{j\}) = p_{\boldsymbol{x}^t}(X^t = \{j\})$ as the probability of product j

[3]Note that $\mathrm{P}(\Omega)$ is finite as Ω is finite by assumption

being purchased at time point t when tuple $\boldsymbol{x}^t$ is offered. Note that p has to satisfy the corresponding Kolmogorov axioms, as it is a probability measure:

$$\begin{aligned} p_{\boldsymbol{x}^t}(j) \geq 0 \quad \forall j \in \{1, \dots n\} \qquad & \text{(non-negativity)} \\ p_{\boldsymbol{x}^t}(\Omega) = 1 \qquad & \text{(unitarity)} \\ p_{\boldsymbol{x}^t}(\cup_{j \in J}\{j\}) = \sum_{j \in J} p_{\boldsymbol{x}^t}(j) \quad \forall J \subset \Omega \qquad & \text{(additivity in discrete setting)} \end{aligned}$$

How to arrive at those probabilities is discussed in Section 2.1.3. Now, we formalize the aim of the firm. The expected revenue-to-go starting with time period t and capacity $\boldsymbol{c}$ is given by the value function $V(t, \boldsymbol{c})$, which maximizes the expected revenue and is mostly formulated recursively because of complicated evolutions due to capacities changing over time.

$$V(t, \boldsymbol{c}) = \max_{x^t \in \{0,1\}^n} \left\{ \sum_{j \in [n]} p_{\boldsymbol{x}^t}(j) \left(r_j + V(t+1, \boldsymbol{c} - \boldsymbol{a}_j)\right) + p_{\boldsymbol{x}^t}(0) V(t+1, \boldsymbol{c}) \right\} \tag{2.1}$$

$$= \max_{x^t \in \{0,1\}^n} \left\{ \sum_{j \in [n]} p_{\boldsymbol{x}^t}(j) \left(r_j - \Delta_j V(t+1, \boldsymbol{c})\right) \right\} + V(t+1, \boldsymbol{c}) \ \forall t \in [T], \boldsymbol{c} \geq 0 \tag{2.2}$$

with $\Delta_j V(t+1, \boldsymbol{c}) \coloneqq V(t+1, \boldsymbol{c}) - V(t+1, \boldsymbol{c} - \boldsymbol{a}_j)$ and boundary conditions $V(T+1, \boldsymbol{c}) = 0$ if $\boldsymbol{c} \geq \boldsymbol{0}$ and $V(t, \boldsymbol{c}) = -\infty \ \forall t \in [T+1]$ if $\boldsymbol{c} \ngeq \boldsymbol{0}$, i. e. $\exists h \in [m] : c_h < 0$.

Note that for determining the optimal offerset, the very last summand $V(t+1, \boldsymbol{c})$ is irrelevant for the maximization as it is constant over all offersets. The optimal offerset $\hat{\boldsymbol{x}}^t$ is thus given by

$$\hat{\boldsymbol{x}}^t = \arg\max_{x^t \in \{0,1\}^n} \left\{ \sum_{j \in [n]} p_{\boldsymbol{x}^t}(j) \left(r_j - \Delta_j V(t+1, \boldsymbol{c})\right) \right\} \ \forall t \in [T], \boldsymbol{c} \geq 0 \tag{2.3}$$

Discrete Choice Model for Customer Behaviour

We already discussed that at each time period at most one customer arrives (out of L customer segments), who can purchase at most one product. The only remaining question is which product is being purchased during a particular time period. Multiple models

exist to describe purchase probabilities[4], but many of them are based on preferences in accordance with utility. We first introduce the utility concept in general and then introduce the multinomial logit model (MNL), which is well known in practice and can be found e. g. in Train (2009). Note that we introduce the concept for a general offerset $\boldsymbol{x}$ in order to easily focus on the important aspects. In reality, and in our implementation, the offerset is considered time-dependent $\boldsymbol{x}^t$.

Consider customer segment l. Let $u_{lj} \in \mathbb{R}_0^+ \ \forall j \in [n]$, resp. $u_{l0} \in \mathbb{R}^+$ denote the utility that a customer of segment l assigns to product j, resp. to the no-purchase alternative. Then, the purchase probability of product j given offer tuple $\boldsymbol{x}$ is described by $p_{l\boldsymbol{x}}(j)$, where again $p_{l\boldsymbol{x}}(0)$ denotes the probability of no-purchase. In MNL, the concrete probabilities are derived by the following formulas

$$p_{l\boldsymbol{x}}(j) = \frac{u_{lj}x_j}{u_{l0} + \sum_{p\in[n]} u_{lp}x_p} \ \forall j \in [n] \quad \text{(probability of purchasing product } j\text{)} \tag{2.4}$$

$$p_{l\boldsymbol{x}}(0) = 1 - \sum_{j\in[n]} p_{l\boldsymbol{x}}(j) \quad \text{(probability of no-purchase)} \tag{2.5}$$

We want to point out a couple of finer details. Note that one often finds the term *consideration set* C_l defined in this setting. It describes the set of all products being considered for purchase by customers of segment l, but this distinction is mathematically irrelevant: Utilities are assigned by the logic $u_{lj} > 0$ if customers of segment l might purchase product j (the higher, the more interested) and $u_{lj} = 0$ if customers are not interested in the product. Thus, the consideration set is simply given by $C_l = \{j \in [n] : u_{lj} > 0\}$ and we could reduce the summations in Equation (2.4) and in Equation (2.5) to go just over C_l as the other summands equal 0 either way. The preference weights of one customer segment can be summarized by the vector $\boldsymbol{u}_l = (u_{l1}, \dots, u_{ln})^\intercal$ and no-purchase preference of u_{l0}. Furthermore, this setting is over-parametrized as described in the following: There are $L + 1$ free parameters, but still one degree of freedom as the two different utility vectors $\hat{\boldsymbol{u}} = 2\boldsymbol{u}$ result in the same probability measures: $\hat{p}_{l\boldsymbol{x}}(j) = \frac{\hat{u}_{lj}x_j}{\hat{u}_{l0}+\sum_{p\in[n]}\hat{u}_{lp}x_p} = \frac{2u_{lj}x_j}{2u_{l0}+\sum_{p\in[n]}2u_{lp}x_p} = \frac{2u_{lj}x_j}{2\left(u_{l0}+\sum_{p\in[n]}u_{lp}x_p\right)} = \frac{u_{lj}x_j}{u_{l0}+\sum_{p\in[n]}u_{lp}x_p} = p_{l\boldsymbol{x}}(j)$. Thus, we can normalize utilities by e. g. setting $u_{l0} = 1$ (which is possible, if $u_{l0} > 0$ is assumed).[5]

[4] Examples for such models are the already mentioned independent demand, lowest open fare or the LC-MNL model.

[5] To give a concrete example: In our example as presented in Figure 2.1d, and therein for customer segment 1, we could change the preference vector to $(2, 4)$ and the no-purchase preference to 1. Ceteris

Together with the uncertainty of which customer segment arrives (if any), we end up at a purchase probability for product j given offerset $\boldsymbol{x}$ of $p_{\boldsymbol{x}}(j)$ and a no-purchase probability of $p_{\boldsymbol{x}}(0)$ as presented in the following. Thus, $p_{\boldsymbol{x}}(j)$ can be interpreted as the deterministic quantity of product j being sold, if set $\boldsymbol{x}$ is offered.

$$p_{\boldsymbol{x}}(j) = \sum_{l \in [L]} \lambda_l p_{l\boldsymbol{x}}(j) \tag{2.6}$$

$$p_{\boldsymbol{x}}(0) = 1 - \sum_{j \in [n]} p_{\boldsymbol{x}}(j) \tag{2.7}$$

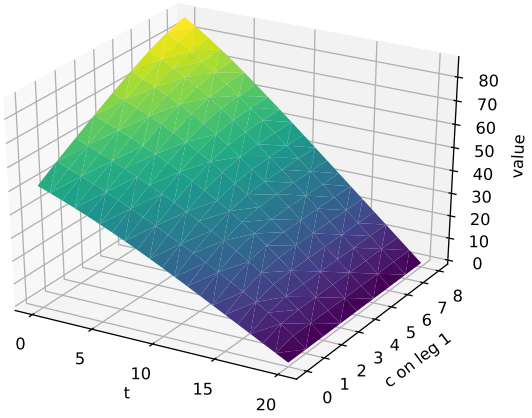
2.2 Exact solution

The first approach to come to ones mind is to maximize the expected value of all revenues to gain given time period t and capacity $\boldsymbol{c}$ directly. So the value function $V(t, \boldsymbol{c})$ is computed recursively starting at the final period T and at all possible capacities and the algorithm then working backwards in time until period 1. We name this approach *Exact Solution* (ES).

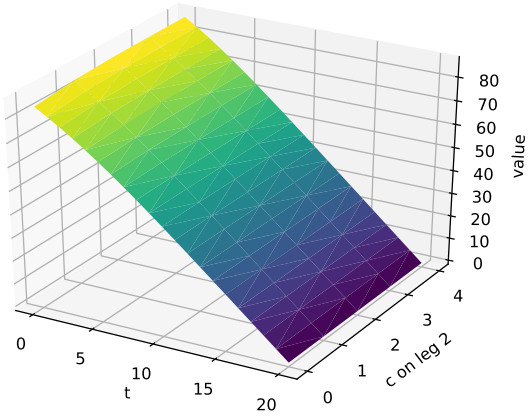
Figure 2.3 presents the resulting value function as three dimensional plots. As we have a total of five variables (four resources and one time) and the associated value of the value function, six dimensions would be needed to depict all the solutions. We reduce the number of dimensions by just allowing one particular resource to change its capacity and fix the capacities of the other resources to its maximum. Note that resource 1 seems to be of particular high value as reducing its capacity to zero diminishes the value function the most, e. g. considering $t = 0$. On the contrary, resource 2 appears to have very little impact on the value function as almost no changes can be recognized for changing values of capacity. All subfigures show the clear dominance of time on the value function as the value function decreases the steepest on the time axis until the expected value of zero at the end of the booking horizon $t = 20$.

ES is difficult to apply in practice, mainly due to two issues as pointed out e. g. in Koch (2017). First, the decision problem inherent at each state (time period t together with capacity $\boldsymbol{c}$) is an assortment optimization problem over 2^n possible offer tuples, i. e. exponential in the total number of products n. Already our tiny working example with 6 products (compare Figure 2.1b) leads to $2^6 = 64$ offer sets. Second, for each time period

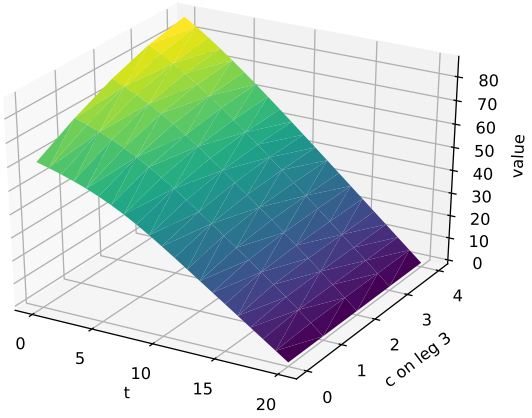
paribus, this would leave the purchase probabilities of customer segment 1 unchanged.



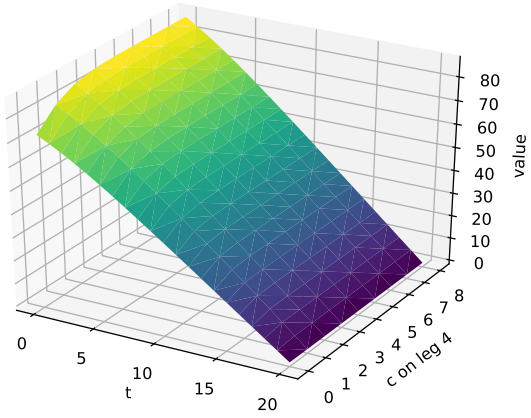
(a) Varying capacity on leg 1.



(b) Varying capacity on leg 2.



(c) Varying capacity on leg 3.



(d) Varying capacity on leg 4.

Figure 2.3: Value function of working example with time $t \in [20]$ on x-axis, capacity as stated on the y-axis (remaining capacities set to their maximal value) and the value depicted on z-axis.

t, the value function (2.1) has to be computed for all possible combinations of capacities. This as well leads to exponential complexity, this time in the total number of resources m. In general, $\prod_{h\in[m]}(c_h^0+1)$ instances have to be solved[6]. Already our tiny working example with 4 resources (compare Figure 2.1c) leads to a total of $9 \cdot 5 \cdot 5 \cdot 9 = 2.025$ instances.

[6]The capacity of each resources can take values in $\{0, 1, \ldots, c_h^0\}$

2.3 Choice based linear programming

Because of the challenges for the exact solution as stated above, approximations are needed to solve Equation (2.1). A popular approach is to approximate the stochastic quantities (purchase probabilities) by deterministic values such as their expected value and to allow capacity and demand to be continuous. This approach is named *Choice based linear programming* (CDLP) and is applied by e. g. Gallego et al. (2004), Liu and van Ryzin (2008) or Bront et al. (2009). In the following, we first introduce the concepts underlying CDLP and its direct implementation in Section 2.3.1. Afterwards, we present two different Column Generation techniques in Section 2.3.2 .

2.3.1 CDLP model and direct implementation

Taking an eagle-eye perspective, the company has to decide which sets to offer in every single time period. But as purchase probabilities don't change over time, the specific time period when to offer an individual set is indistinguishable. Thus, we can aggregate over time. We introduce a few more notation to put this idea into formulas. Considering one time period: As already stated in Section 2.1.3, $p_{\boldsymbol{x}}(j)$ can be interpreted as the deterministic quantity of product j being sold. Let $R(\boldsymbol{x})$ represent the expected (thus deterministic) revenue given offer set $\boldsymbol{x}$, and let $\boldsymbol{Q}(\boldsymbol{x}) = (Q_1(\boldsymbol{x}), \ldots, Q_m(\boldsymbol{x}))^\intercal$ denote the expected capacity being used, i. e.

$$R(\boldsymbol{x}) = \sum_{j \in [n]} r_j p_{\boldsymbol{x}}(j) \tag{2.8}$$

$$Q_h(\boldsymbol{x}) = \sum_{j \in [n]} a_{hj} p_{\boldsymbol{x}}(j) \ \forall h \in [m] \tag{2.9}$$

Note that before, we considered just one time period. Now, we aggregate over time periods and denote the total time[7] for which set $\boldsymbol{x}$ is offered by $t(\boldsymbol{x})$ and the set of all possible offer tuples as $N = \{0,1\}^n$. Thus, we can formulate the choice-based linear program $V^{CDLP}(T, \boldsymbol{c})$.[8] The optimization problem is solved by varying the decision variables $t(\boldsymbol{x})$.

[7] Total time meaning number of time periods.

[8] Note that in our formulation, $p_{\boldsymbol{x}}(j)$ already includes the arrival rate λ and thus, λ doesn't appear directly in our formulation of CDLP in contrast to the formulation in Bront et al. (2009).

$$V^{CDLP}(T, \boldsymbol{c}): \quad \max \sum_{\boldsymbol{x} \in N} R(\boldsymbol{x}) t(\boldsymbol{x}) \tag{2.10}$$

$$\text{s.t.} \sum_{\boldsymbol{x} \in N} Q_h(\boldsymbol{x}) t(\boldsymbol{x}) \leq c_h \ \forall h \in [m] \tag{2.11}$$

$$\sum_{\boldsymbol{x} \in N} t(\boldsymbol{x}) \leq T \tag{2.12}$$

$$t(\boldsymbol{x}) \geq 0 \ \forall \boldsymbol{x} \in N \tag{2.13}$$

We want to mention some characteristics of this optimization problem. By allowing $t(\boldsymbol{x})$ to be continuous[9], we arrive at a classical *linear programming problem* (LP), for which efficient solution methods exist (such as Simplex) and are implemented (e. g. in Gurobi). Even though the number of variables is exponential (2^n as $N = \{0, 1\}^n$), methods such as column-generation techniques can be used to solve this LP efficiently and are introduced in Section 2.3.2.[10] We now use concepts of duality theory to lead the way to column generation techniques.

For a start, note that the optimization problem $V^{CDLP}(T, \boldsymbol{c})$ has $m + 1$ constraints - m in Constraint (2.11) and 1 in Constraint (2.12). By duality theory and especially Complementary Slackness, compare e. g. to Domschke et al. (2015), at most $m + 1$ basic variables will end up being non-zero. Hence, even though the number of variables being exponentially large (2^n), the solution will comprise at most a linear number ($m + 1$) of variables, which again motivates the usage of column-generation techniques.

The dual of the CDLP is given by

$$VD^{CDLP}(T, \boldsymbol{c}): \quad \min \boldsymbol{\pi}^\intercal \boldsymbol{c} + \sigma T \tag{2.14}$$

$$\text{s.t.} \ \boldsymbol{\pi}^\intercal \boldsymbol{Q}(\boldsymbol{x}) + \sigma \geq R(\boldsymbol{x}) \ \forall \boldsymbol{x} \in N \tag{2.15}$$

$$\boldsymbol{\pi} \geq 0 \tag{2.16}$$

$$\sigma \geq 0 \tag{2.17}$$

where $\boldsymbol{\pi} \in \mathbb{R}^m$ is the vector of dual variables corresponding to the resource constraints (2.11) and σ is the dual variable corresponding to the time constraint (2.12). Again with

[9] E. g. $t(\boldsymbol{x}) = 1.2$ corresponds to offerset $\boldsymbol{x}$ being offered for one whole time period and 20 % of another time period.

[10] Note that Gurobi is high-level and thus programmed in such way so that it realizes which method shall be used to solve a given LP efficiently.

theory of duality in mind, the optimal value of π_h estimates the marginal value of resource h and the optimal value of σ provides a marginal value of time, i. e. one additional time period would lead to an expected increase in revenue of σ units.

In our working example, there are $256 = 2^8$ offersets and it is optimal to offer the set of products $\{1, 4, 5\}$ in each time period. More details on the optimal solution are presented in Figure 2.4.

```
Optimize a model with 5 rows, 64 columns and 216 nonzeros
Coefficient statistics:
  Matrix range     [6e-02, 1e+00]
  Objective range  [7e-01, 5e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+00, 2e+01]
Presolve removed 0 rows and 33 columns
Presolve time: 0.00s
Presolved: 5 rows, 31 columns, 107 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    4.6288150e+02   6.226968e+01   0.000000e+00      0s
       3    9.1952381e+01   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.00 seconds
Optimal objective  9.195238095e+01


------------------------------------------------------------


Optimal objective value:
 91.95238095238096

Optimal solution:
Tuple (1, 0, 0, 1, 1, 0)  is offered for a total time of  20.0


Product  1 is offered during  20.0000  time periods.
Product  2 is offered during   0.0000  time periods.
Product  3 is offered during   0.0000  time periods.
Product  4 is offered during  20.0000  time periods.
Product  5 is offered during  20.0000  time periods.
Product  6 is offered during   0.0000  time periods.
```

Figure 2.4: Optimal solution with the null set present at start.

2.3.2 Solving CDLP by column generation

The huge problem size, note again the 2^n primal variables in Equation (2.10), might cause problems. But this can be taken care of by column generation techniques as done by e. g. Gallego et al. (2004), Liu and van Ryzin (2008) or Bront et al. (2009). We first sketch the algorithm, then elaborate on its complexity and finally present approaches for solving the problem.

CDLP column generation algorithm

We shortly present the idea of CDLP and will elaborate in the details in the following paragraphs. The idea to handle the exponential problem size lies in starting with a small number (e. g. one) of initial offer tuples and incrementally adding promising offer tuples. We start with one offer tuple $\boldsymbol{x}_1$[11], i. e. only one primal variable (instead of N) and solve a reduced linear program using only this column. After checking for the existence of any other offersets with positive reduced cost relative to the current dual prices of the reduced problem, we add a corresponding positive reduced cost primal variable and re-optimize the LP. If no offerset with positive reduced cost exists, the current solution of the problem is optimal.

The reduced primal CDLP problem at step k, i. e. with k distinct offersets included in the set $\mathcal{N} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, is given by

$$
\begin{aligned}
V^{CDLP-R}(T, \boldsymbol{c}): \qquad & \max \sum_{\boldsymbol{x} \in \mathcal{N}} R(\boldsymbol{x}) t(\boldsymbol{x}) && (2.18)\\
\text{s.t.} \quad & \sum_{\boldsymbol{x} \in \mathcal{N}} Q_h(\boldsymbol{x}) t(\boldsymbol{x}) \leq c_h \ \forall h \in [m] && (2.19)\\
& \sum_{\boldsymbol{x} \in \mathcal{N}} t(\boldsymbol{x}) \leq T && (2.20)\\
& t(\boldsymbol{x}) \geq 0 \ \forall \boldsymbol{x} \in \mathcal{N} && (2.21)
\end{aligned}
$$

Again, with $\boldsymbol{\pi} \in \mathbb{R}^m$, resp. $\sigma \in \mathbb{R}$ denoting the dual variables of resources, resp. time, the corresponding dual problem is given by

[11] In alignment with Bront et al. (2009), this offerset comprises the first product of interest of each customer segment.

$$VD^{CDLP-R}(T, \boldsymbol{c}): \quad \min \boldsymbol{\pi}^\intercal \boldsymbol{c} + \sigma T \tag{2.22}$$

$$\text{s.t. } \boldsymbol{\pi}^\intercal \boldsymbol{Q}(\boldsymbol{x}) + \sigma \geq R(\boldsymbol{x}) \ \forall \boldsymbol{x} \in \mathcal{N} \tag{2.23}$$

$$\boldsymbol{\pi} \geq 0 \tag{2.24}$$

$$\sigma \geq 0 \tag{2.25}$$

Let us analyse $V^{CDLP-R}(T, \boldsymbol{c})$ in more depth to explore the ideas behind column generation algorithms. Equation (2.22) shall be minimized by altering the (dual) variables $\boldsymbol{\pi}$ and σ. These variables are bounded from below by zero and have to satisfy Constraint (2.23). At this point, we introduce the reduced costs of an offerset $\boldsymbol{x}$ as

$$rc(\boldsymbol{x}) \coloneqq R(\boldsymbol{x}) - \boldsymbol{\pi}^\intercal \boldsymbol{Q}(\boldsymbol{x}) - \sigma$$

and realize that Constraint (2.23) is equivalent to $rc(\boldsymbol{x}) \leq 0$, i. e. all offersets in $\mathcal{N}$ are ensured to have negative reduced costs (the dual variables are chosen in such manner). Remember $N = \{0, 1\}^n$, thus we skipped all offersets $\boldsymbol{x} \in N \backslash \mathcal{N}$. If an offerset $\hat{\boldsymbol{x}}$ currently has negative reduced costs, adding $\hat{\boldsymbol{x}}$ to $\mathcal{N}$ results in one additional line of Constraint (2.23), which is inactive, i. e. satisfied (we added variable $\hat{\boldsymbol{x}}$ with negative reduced costs). Thus, the objective value would not change as the additional constraint does not put a restriction on the optimal dual variables $\boldsymbol{\pi}$ and σ. On the contrary, when considering an offerset $\check{\boldsymbol{x}}$ with positive reduced costs, adding the corresponding constraint in the dual problem will result in a violation of Constraint (2.23). Thus, the corresponding constraint is active and $\boldsymbol{\pi}$ and σ have to be adjusted to become admissible again.

Having this idea in mind motivates the column generation subproblem:[12]

$$\max_{\boldsymbol{x} \in N} rc(\boldsymbol{x}) = \max_{\boldsymbol{x} \in N} \left(R(\boldsymbol{x}) - \boldsymbol{\pi}^\intercal \boldsymbol{Q}(\boldsymbol{x}) \right) - \sigma$$

If the optimal solution to Problem (2.3.2) is positive, we augment $\mathcal{N}$ with the corresponding $\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x} \in N} rc(\boldsymbol{x})$. Note once again, $rc(\hat{\boldsymbol{x}}) > 0$ and thus $\hat{\boldsymbol{x}}$ is not part of the old $\mathcal{N}$.

If the optimal solution to Section 2.3.2 is non-positive, the current solution is already op-

[12] Note that in comparison to Bront et al. (2009), the potential offersets to consider can be limited to $N \backslash \mathcal{N}$ as $\boldsymbol{\pi}$ and σ are chosen in such manner that $rc(\boldsymbol{x}) \leq 0$ is ensured $\forall \boldsymbol{x} \in \mathcal{N}$.

timal. This shall be elaborated in the following. $V^{CDLP}(T, \boldsymbol{c})$ differs from $V^{CDLP-R}(T, \boldsymbol{c})$ only in the considered set of offersets, all N versus the reduced $\mathcal{N}$. We augmented $\mathcal{N}$ to a point, when the column generation subproblem (2.3.2) results in a non-positive solution. Now, Constraint (2.23) is satisfied $\forall \boldsymbol{x} \in \mathcal{N}$ (by construction), but also $\forall \boldsymbol{x} \in N$ (compare to the optimal solution of the column generation subproblem). In summary, the optimal value of $VD^{CDLP-R}(T, \boldsymbol{c})$ equals the optimal value of $VD^{CDLP}(T, \boldsymbol{c})$. Furthermore, $VD^{CDLP}(T, \boldsymbol{c})$ equals $V^{CDLP}(T, \boldsymbol{c})$ as strong duality holds because we are dealing with a linear optimization problem and have found a feasible solution for the dual, which is a sufficient condition for strong duality as proven in Theorem 6.1.7*b*) of Gritzmann (2013).

Note that, up to this point, we have not considered a particular customer choice model. Let us continue with introducing the MNL choice model also in the CDLP context, in order to analyse its complexity in the following. With MNL as customer choice model, Section 2.3.2 can be expressed as (all equivalent)

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \sum_{j\in[n]} r_j p_{\boldsymbol{x}}(j) - \sum_{h\in[m]} a_{hj}\pi_h p_{\boldsymbol{x}}(j) \right\} - \sigma$$

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \sum_{j\in[n]} \left(r_j - A_j^\intercal \boldsymbol{\pi}\right) p_{\boldsymbol{x}}(j) \right\} - \sigma$$

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \sum_{l\in[L]} \lambda_l \frac{\sum_{j\in[n]} \left(r_j - A_j^\intercal \boldsymbol{\pi}\right) u_{lj} x_j}{\sum_{j\in[n]} u_{lj} x_j + u_{l0}} \right\} - \sigma$$

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \left\{ \sum_{j\in[n]} \left(r_j - A_j^\intercal \boldsymbol{\pi}\right) x_j \left(\sum_{l\in[L]} \frac{\lambda_l u_{lj}}{\sum_{j\in[n]} u_{lj} x_j + u_{l0}} \right) \right\} - \sigma$$

Note that the assumptions $u_{lj} \in \mathbb{R}_0^+ \ \forall j \in [n]$, resp. $u_{l0} \in \mathbb{R}^+$ ensure the optimization problem to be well defined.

Complexity of column generation

As noted in Bront et al. (2009), Section 2.3.2 is in general not separable in the variables $x_j,\ j \in [n]$. Multiple customer segments (e. g. l_1 and l_2) might be interested in the same product j and thus, x_j appears in the denominator belonging to l_1 and in the one belonging to l_2. Our column generation problem is part of the general group of *hyperbolic* (or

fractional) *0-1 programming problems*, which were studied by e. g. Hansen et al. (1991) or Borrero et al. (2017). The general version[13] is given by

$$\max_{\boldsymbol{x}\in\{0,1\}^n} \sum_{l\in[L]} \frac{a_{l0} + \sum_{j\in[n]} a_{lj}x_j}{b_{l0} + \sum_{j\in[n]} b_{lj}x_j} \ , \tag{2.26}$$

with no restrictions on a_{lj}, b_{lj} and $x_j \in \{0,1\}\ \forall j \in [n]$. Prokopyev et al. (2005b) have proven that Equation (2.26) is NP-hard by providing a polynomial reduction from the weighted 2SAT problem.

Bront et al. (2009) point out that due to the tight linkage of variables in our column generation subproblem, the general Equation (2.26) cannot be easily reduced to the specific Section 2.3.2. But they reduce the *minimum vertex cover problem* to our problem Section 2.3.2. And as Garey and Johnson (2009) proofs the *minimum vertex cover problem* to be NP-hard[14], our column generation problem is NP-hard as well.

Approaches for solving column generation

Despite the previous theoretical result, two approaches for solving column generation proved themselves valuable in practice and are presented in the following.

MIP Formulation

We follow Bront et al. (2009) and formulate the column generation problem Section 2.3.2 as *mixed integer problem* (MIP). We face two causes of non-linearity: a fraction and a product. To eliminate the fraction, we introduce the variable

$$y_l = \frac{1}{\sum_{j\in[n]} u_{lj}x_j + u_{l0}}$$

and enforce this to hold via the (non-fractional) Constraint (2.28). By also changing the order of summation[15], using the distributive law and ignoring the constant σ (we check if the optimal result is $> \sigma$ in the end), we rewrite the column generation problem Section 2.3.2

[13] Note that $max(x) = -min(-x)$ if the optimium exists. This holds in general, thus it doesn't matter whether we look at minimization or maximization.

[14] Stated in [GT1] Vertex Cover under *A1.1 Covering and Partitioning* on page 190 and referred to Karp (2010), who proves NP-hardness by a transformation from 3SAT, the well established example for a problem of the NP-class.

[15] Changing the order of summation doesn't change the summation value since the sum is finite.

as[16]

$$\max_{\boldsymbol{x},\boldsymbol{y}} \sum_{l\in[L]} \sum_{j\in[n]} (r_j - A_j^\intercal \boldsymbol{\pi}) x_j \lambda_l u_{lj} y_l \tag{2.27}$$

$$\text{s.t. } y_l u_{l0} + \sum_{j\in[n]} u_{lj} x_j y_l = 1 \ \forall l \in [L] \tag{2.28}$$

$$x_j \in \{0,1\} \ \forall j \in [n] \tag{2.29}$$

$$y_l \geq 0 \ \forall l \in [L] \tag{2.30}$$

To eliminate the product $x_j y_l$, we follow the concept of linearizing xy with $x \in \{0,1\}$ and $y \geq 0$ as presented by Klein (1718). We introduce an auxiliary variable $z \geq 0$. Obviously, we want $z = 0$ if $x = 0$. This leads to the constraint $z \leq xy$. And we want $z = y$ if $x = 1$. This can be enforced by putting the constraints $z \leq y$[17] and $z \geq y - M(1-x)$ with M large enough.[18] Note that the other constraints are trivially fulfilled if $x = 1$, resp. $x = 0$. We apply these ideas and introduce the auxiliary variable $z_{lj} = x_j y_l$ with the corresponding constraints to obtain the MIP formulation

$$\max_{\boldsymbol{x},\boldsymbol{y}} \sum_{l\in[L]} \sum_{j\in[n]} (r_j - A_j^\intercal \boldsymbol{\pi}) \lambda_l u_{lj} z_{lj} \tag{2.31}$$

$$\text{s.t. } y_l u_{l0} + \sum_{j\in[n]} u_{lj} z_{lj} = 1 \ \forall l \in [L] \tag{2.32}$$

$$y_l - z_{lj} \leq M(1 - x_j) \ \forall l \in [L] \, \forall j \in [n] \tag{2.33}$$

$$z_{lj} \leq y_l x_j \ \forall l \in [L] \, \forall j \in [n] \tag{2.34}$$

$$x_j \in \{0,1\} \ \forall j \in [n] \tag{2.35}$$

$$y_l \geq 0 \ \forall l \in [L] \tag{2.36}$$

$$z_{lj} \geq 0 \ \forall l \in [L] \, \forall j \in [n] \tag{2.37}$$

[16] Note that Bront et al. (2009) used sloppy notation by claiming $j \in N$, as she introduced N to be a constant (the total number of products). Our notation is precise.

[17] Note that this constraint is already included in the previous one $z \leq yx$, thus can be excluded here. This thought can be used to enhance Klein (1718) by combining constraints (1.20) and (1.21) to $\lambda_{ij} \leq p_j x_{ij}$. Note that constraint (1.20) does not really come from enforcing the linearization, but from the participation constraint (1.14).

[18] This idea of introducing a large constant to switch off a constraint if need be is generally referred to as *big M*.

with $M \geq 1/\min\{u_{l0} : l \in [L]\}$[19]. M is large enough, because Constraint (2.33) is satisfied if activated ($x_j = 0$) as z_{lj} is then forced to zero by Constraint (2.34) and y_l potentially set to $1/u_{l0}$ by Equation (2.28).

Having such a MIP formulation allows for the usage of standard MIP solvers like Gurobi. But as the complexity result of Bront et al. (2009) states, the column generation problem is still NP hard. Thus, we present a polynomial time heuristic.

Greedy Heuristic

To identify the most promising offerset to add to the current set of offersets $\mathcal{N}$, we use a greedy, constructive heuristic. It was suggested by Bront et al. (2009), with an underlying algorithm originally proposed by Prokopyev et al. (2005a) to solve a general fractional programming problem. We adapt notation heavily to present a more concise and mathematical version. The algorithm makes straightforward use of the most promising offersets. Note that, via the current offersets to consider $\mathcal{N}$, we arrived at values of the dual variable $\boldsymbol{\pi}$ (and σ, but this is not needed).

The heuristic itself is given in Algorithm 1[20] and is explained in the following. The algorithm starts in Line 1 with defining the value of an offerset candidate X. This just rewrites the column generation subproblem Section 2.3.2 by directly summing solely over the products in the offerset candidate instead of summing over all products and just keeping those for which $x_j = 1$. Line 2 initializes the set final set S, which will comprise the offerset in the end, and the set S', which contains promising products, i. e. such products with positive reduced costs. Line 3 provides a fail-safe: If no product is reasonable (too high opportunity costs), we immediately return the empty set to be offered. In Line 4, the most promising product j^* is selected. Within the loop, sets S and S' are updated (Line 6) and the next suitable product j^* is determined. Whenever including this product j^* in the offerset S makes sense, i. e. raising the value of the offerset, the loop iterates again. When the value function does not increase further, the optimal offerset S is returned.

[19] In comparison to Bront et al. (2009), we adapt the minimum as not all products might be in the consideration set, thus $\underline{v}$ might be 0. Our denominator will always be nonzero, as all u_{l0} are assumed to be > 0, and will be appropriate as potentially adding some positive u_{lj} just increases the denominator, thus decreases M.

[20] Note that our first j^*, found in Line 4, is in consistence with the other j^*'s found in Line 7, i. e. including the arrival probabilities λ_l. This is not the case in Bront et al. (2009), compare step 3 with step 4.a.

1: $\texttt{Value}(X) \coloneqq \sum_{l=1}^{L} \lambda_l \frac{\sum_{i \in X}(r_i - A_i^\intercal \pi) u_{li}}{\sum_{i \in X} u_{li} + u_{l0}}$
2: $S \coloneqq \emptyset, \quad S' \coloneqq \left\{ j \in [n] : r_j - A_j^\intercal \pi > 0 \right\}$
3: **if** $S' = \emptyset$ **then return** $\emptyset$
4: $j^* \coloneqq \arg\max_{j \in S'} \texttt{Value}(\{j\})$
5: **repeat**
6: $\quad S \coloneqq S \cup \{j^*\}, \quad S' \coloneqq S' \backslash \{j^*\}$
7: $\quad j^* \coloneqq \arg\max_{j \in S'} \texttt{Value}(S \cup \{j\})$
8: **until** $\texttt{Value}(S \cup \{j^*\}) \leq \texttt{Value}(S)$
9: **return** S, e. g. as offerset $\boldsymbol{x}$ given by $x_j = \mathbb{1}_{\{j \in S\}}, \ j \in [n]$

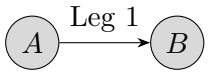
Algorithm 1: Greedy Heuristic

This heuristic has worst-case complexity of $O(n^2 L)$ as the repeat loop might run for all n products and in Line 7 the calculation of the arg max of `Value` is done $|S'|$ times with a summation over all L customer segments and the remaining, $n - |S'|$ products. Note that in practice, there are much less customer segments than products, i. e. $L << n$, leading to less then cubic complexity in n.

We want to present an adapted version[21] of the example in Bront et al. (2009) to show that Algorithm 1 is indeed a heuristic, i. e. stops without finding the optimal solution. We also use this example as first validation of our code. Consider the column generation subproblem Section 2.3.2 with $n = 3$, $L = 3$, other parameters as specified in Figure 2.5 and $\boldsymbol{\pi} = (0, 0, 0)^\intercal$. Applying our implemented functions result in Figure 2.6, which proofs that the greedy heuristic was unable to find the optimum as it stopped at the optimal value of 16.66 instead of arriving at the truly optimal 17.83.

[21] We adjust parameters $\lambda_l = 1/3$ to sum up to 1. Note, this is necessary due to our assumption of at most one customer arriving at a given point in time. To reproduce the optimal values stated in Bront et al. (2009), we multiply the objective value by 3 due to the scalability of Poisson processes as stated in Lemma 14.

(a) Airline network.



(b) Products.

Product	Origin-destination	Fare
1	$A \to B$	100
2	$A \to B$	19
3	$A \to B$	19

(c) Resources.

Leg	Capacity
1	∞

(d) Customers.

Seg.	λ_l	Consideration tuple	Preference vector	No purchase preference	Description
1	1/3	$(1, 2, 3)$	$(1, 1, 1)$	1	Flexible, (A→B)
2	1/3	(2)	(1)	1	Just 2, (A→B)
3	1/3	(3)	(1)	1	Just 3, (A→B)

Figure 2.5: Small example for illustration of sub-optimality of CDLP Greedy Heuristic with $T = 1$ time period.

```
MIP results in: optimal value =  17.83    optimal tuple =  (1, 1, 1)
GH results in:  optimal value =  16.66    optimal tuple =  (1, 0, 0)
```

Figure 2.6: Results for small example. MIP for "CDLP with MIP formualtion" vs. GH for "CDLP with greedy heuristic".

Our final implementation of the CDLP by column generation consists of: At first, we use the greedy heuristic to identify a promising offerset. If this method does not succeed, we use the exact MIP procedure. If no new offerset is identified to enter the base, we have found the optimal solution of the CDLP. Note that we include the empty set in the set of products to consider, which ensures (compare Constraint 2.20) that during the whole booking horizon a set is offered.

Applying this version of CDLP by column generation with the greedy heuristic to our working example, we arrive at the optimal solution of offering set $\{1, 4, 5\}$ over the whole time horizon, as presented in Figure 2.7. Note that the same solution is found as by using CDLP in its MIP formulation, compare to Figure 2.4. But the column generation version with the greedy heuristic needs just 3 columns in comparison to 256 used by the MIP version.

```
Optimal objective value:
 91.95238095238096

Found by using  3  columns.

Optimal solution:
Tuple (1, 0, 0, 1, 1, 0)  is offered for a total time of  20.0


Product  1 is offered during  20.0000  time periods.
Product  2 is offered during   0.0000  time periods.
Product  3 is offered during   0.0000  time periods.
Product  4 is offered during  20.0000  time periods.
Product  5 is offered during  20.0000  time periods.
Product  6 is offered during   0.0000  time periods.
```

Figure 2.7: Optimal solution for CDLP by column generation for working example.

Chapter 3

Approximate dynamic programming

We now move on to a new solution approach, namely approximate dynamic programming. The method is first motivated and put in context of the thesis. Secondly, a short classification of different methods belonging to this class is presented, before the method applied in our setting is discussed in detail.

ADP builds upon Equation (2.3). The real challenge is to determine the opportunity costs per product j, i. e. $\Delta_j V(t+1, \boldsymbol{c})$. ADP approximates these opportunity costs additively using bid prices for the resources h, $\pi_h(t, c_h)$, if those are needed to produce product j, i. e. $\Delta_j V(t+1, \boldsymbol{c}) = \sum_{h\in[m]} a_{hj}\pi_h(t+1, c_h)$. Thus, the policy is given as the solution of

$$ {}^{\pi}\boldsymbol{x}^t = \underset{\boldsymbol{x}^t\in\{0,1\}^n}{\arg\max} \left\{ \sum_{j\in[n]} p_j(\boldsymbol{x}^t) \left(r_j - \sum_{h\in[m]} a_{hj}\pi_h(t+1, c_h) \right) \right\} , \tag{3.1} $$

and is therefore completely described by the bid prices $\pi_h(t, c_h)$ $\forall t \in [T]$. Note that $\boldsymbol{x}^t$ represents the set offered during time period t, i. e. the decision is made at time point $t-1$. $\Delta_j V(t+1, \boldsymbol{c})$ represents the difference in revenue to go when looking at selling product j and considering time periods $t+1, \ldots, T$ and capacities $\boldsymbol{c}$. Note that when considering the very last time period T, opportunity costs $\Delta_j V(t+1, \boldsymbol{c})$ are zero $\forall j \in [n], \forall \boldsymbol{c} \geq 0$.

We now present a brief overview of how to classify simulation based ADP methods, which the textbooks Bertsekas (2005) and Powell (2011) cover in more detail. The methods are named *approximate*, as they are based on approximating the value function. Various sample paths are generated and used in two different ways to iteratively improve the approximation of the value function: either by *approximate value iteration* (AVI) or by *approximate policy iteration* (API). Each iteration of AVI comprises a single sample path. It evaluates the current policy on this particular sample path and updates parameters for

the value function approximation after each period of the sample path. On the contrary, each iteration of API comprises multiple sample paths. It evaluates the current policy on a set of sample paths and uses the combined information to update and improve the value function approximation (and thus the policy). Thus, one iteration of API is certainly costlier and usually API requires less iterations to obtain good policies. One obvious prerequisite for this is a large enough set of sample paths. We refer to Powell (2011) for more details.

A suitable method to be used for our setting was proposed by Koch (2017) and belongs to the class of simulation based ADP. More precisely, it comprises a value function approximation algorithm and uses policy iteration. It approximates the value function $V(t, \boldsymbol{c})$ by computationally simple functions (linear or piecewise linear), makes use of an offline phase to determine bid prices $\pi_h(t, c_h)\ \forall t \in [T]$ and applies Equation (3.1) in an online phase to compute the offerset.

3.1 Approximate policy iteration

Approximate Policy Iteration is used in the offline phase and aims at determining bid prices for each time period $\pi_h(t, c_h)\ \forall t \in [T]$. These bid prices fully determine the policy and the final bid prices are then used in the online phase. The complete algorithm is given by Figure 3.1 and comprises several steps. We first give an overview of the concept and then explain the details step by step.

3.1.1 API value function approximation

We now introduce the concept of the method in the following and created Table 3.1 for reference, as there are quite a number of variables involved in API.

Symbol	Domain	Meaning
θ_t	$\mathbb{R}$	optimization variable (offset) for time period t
Θ	$\mathbb{R}^\intercal$	optimization variable comprising all θ_t
$\boldsymbol{\pi}_t$	$\mathbb{R}^m$	optimization variable (bid price for each resource) for time period t
$\boldsymbol{\Pi}$	$\mathbb{R}^{m \times T}$	optimization variable (bid price for each resource) comprising all $\boldsymbol{\pi}_t$
$\hat{V}_t^i$	$\mathbb{R}$	sample revenue to go for sample i starting at and including period t
$\hat{V}$	$\mathbb{R}^{T \times I}$	all sample revenues to go comprising all $\hat{V}_t^i$
$\hat{\boldsymbol{C}}_{t-1}^i$	$\mathbb{R}^m$	available capacities for resources for sample i at end of period $t-1$
$\hat{\boldsymbol{C}}$	$\mathbb{R}^{m \times T \times I}$	all available capacities comprising all $\hat{\boldsymbol{C}}_{t-1}^i$
r_t	$\mathbb{R}$	sample revenue generated during time period t
$\boldsymbol{c}$	$\mathbb{N}^m$	available capacities for each resource at current time period
$\boldsymbol{c}^0$	$\mathbb{N}^m$	starting capacities
$\boldsymbol{x}$	$\{0,1\}^n$	offerset at current time period
ϵ_t	$[0,1]$	epsilon used at time period t

Table 3.1: API overview of parameters.

Set $\theta_t = 0$ and $\boldsymbol{\pi}_t = \mathbf{0}\ \forall t \in [T]$
for `k = 1 to K` **do**
  Set $\hat{V}_t^i = 0$ and $\hat{\boldsymbol{C}}_{t-1}^i = 0\ \forall t \in [T], \forall i \in [I]$
  for `i = 1 to I` **do**
    Set $\hat{r}_t = 0$ and $\hat{\boldsymbol{c}}_{t-1} = 0\ \forall t \in [T]$
    Initialize $\boldsymbol{c} = \boldsymbol{c}^0$
    for `t = 1 to T` **do**
      $\hat{\boldsymbol{c}}_{t-1} \coloneqq \boldsymbol{c}$
      Get $\boldsymbol{\pi}(t, \boldsymbol{c})$
      Compute $\boldsymbol{x} =$ `determineOfferset`$(\boldsymbol{\pi}(t, \boldsymbol{c}), \epsilon_t)$
      Simulate a sales event $j' \in \{0, 1, \ldots, n\}$
      if $j' \in \{1, \ldots, n\}$ **then**
        $\hat{r}_t = r_{j'}$ and $\boldsymbol{c} = \boldsymbol{c} - \boldsymbol{a}_{j'}$
    Compute $\hat{V}_t^i = \sum_{\tau=t}^{T} \hat{r}_t\ \forall t \in [T]$
    Assign $\hat{\boldsymbol{C}}_{t-1}^i = \hat{\boldsymbol{c}}_{t-1}\ \forall t \in [T]$
  $(\Theta, \boldsymbol{\Pi}) =$ `updateParameters`$(\hat{V}, \hat{\boldsymbol{C}}, \Theta, \boldsymbol{\Pi}, k)$
return $(\Theta, \boldsymbol{\Pi})$

Figure 3.1: Approximate policy iteration. Note that t represents a time period and therefore takes values in $[T]$. As at time period t, there is c_{t-1} capacity available, we use a different index for capacity.

API aims at approximating the value function for each state. A state is characterized by current time period t and vector of current capacities $\boldsymbol{c}$. Therefore, a natural choice is to use a linear function as given by

$$V_{t,\boldsymbol{c}}(\theta_t, \boldsymbol{\pi}_t) \coloneqq \theta_t + \sum_{h \in [m]} \pi_{th} c_h \tag{3.2}$$

with

$$\theta_t \geq 0 \tag{3.3}$$

$$\max_{j=1,\dots,n} r_j \geq \pi_{th} \geq 0 \tag{3.4}$$

and $\theta_{T+1} = 0$ as well as $\pi_{T+1,h} = 0 \; \forall h \in [m]$ assumed implicitly, thus leading to $V(T+1, \boldsymbol{c}) = 0$ if $\boldsymbol{c} \geq \boldsymbol{0}$. Note that we follow Koch (2017) and include expert knowledge on the problem. The non-negativity constraints in Equation (3.3) and Equation (3.4) ensure that the optimal value is non-negative and increasing in capacity. The upper bound in Equation (3.4) is also reasonable as one additional capacity should increase the optimal value less than the amount of revenue generated by selling the most expensive product. This directly leads us to the interpretation of the variables. θ_t represents the constant offset. π_{th} can be directly interpreted as bid prices of the resources, and therefore be used in Equation (3.1): $\pi_h(t, c_h) = \pi_{th}$. Note that in this setting, the bid price is independent of the current level of capacity.[1]

3.1.2 API overview

Let us now discuss the algorithm in depth.[2] For the first policy iteration, Line 1 sets all optimization variables to zero which results in the greedy policy of offering the set resulting in the highest expected revenue (no consideration of costs) in each time period.

Then, a total of K policy iterations follow.[3] In each policy iteration k, the current policy

[1] In comparison of a linear with a piecewise linear approach.

[2] We strictly use our notion of time periods in this thesis. Note that there is a one-to-one correspondence of "action happening during time period t with $t \in [T]$" and "action happening at time point t with $t \in \{0, \dots, T-1\}$". The latter is more convenient to use in the Python implementation as Python indexing starts at 0.

[3] Note that this direct approach is used in Koch (2017). While a suitable K was presumably found looking at convergence rates and then fixed, I would encourage practitioners to directly apply a convergence criterion for termination of policy iteration. Note that policy iteration is proven to converge if a discounting factor of later revenues is included. Compare Bertsekas(2005; Proposition 1.3.6 on p. 48).

is evaluated first. To do so, we use the current policy to evaluate I random sample paths. Line 3 sets up the dataset for revenues to go $\hat{V}_t^i$ and remaining capacities $\hat{\boldsymbol{C}}_{t-1}^i$ for all time periods $t \in [T]$ and sample paths $i \in [I]$.

For each sample path i, a storage unit for revenue $\hat{r}_t$ is created for all time periods $t \in [T]$ and all capacities $\hat{\boldsymbol{c}}_{t-1}$ (Line 5). Note that we use a different index for capacities to account for the fact that for the remaining revenues to go starting from and including time period t, we have capacities $\boldsymbol{c}_{t-1}$ available (those being left over from time period $t-1$.). Furthermore, the capacity is set to the initial capacity in Line 6.

For each time period, the available capacity $\hat{\boldsymbol{c}}_{t-1}$ (left over at end of previous time period) is stored in Line 8, the current bid prices are calculated in Line 9 and used to determine the offerset $\boldsymbol{x}$ in Line 10. More details on the determination of the offerset can be found in Section 3.1.3. With this information, a sales event j' is simulated and in case a product has been sold ($j' \in [I]$), the revenue is stored and capacity adjusted accordingly. Note that the revenue thus belongs to time period t and capacity will affect time period $t+1$ (Line 11 to Line 13).

After simulating one sample path, the value function for this sample path $\hat{V}_t$ is evaluated for all time periods $t \in [T]$ in Line 14, and so are the capacities in Line 15.

Finally, Line 16 comprises the policy improvement step and uses all the information of the current policy iteration to update the optimization variables $\theta_t, \boldsymbol{\pi}_t \; \forall t$. Before expanding on how this update is executed, we want to fill the gap on how the offerset is determined in Line 10.

3.1.3 API determination of offersets

Our goal is to efficiently determine a reasonable set of products to offer. Thus, we use a heuristic and reduce the amount of products to consider as fast as possible. The following algorithm is based on the ideas of the greedy, largest marginal benefit heuristic for the column generation subproblem outlined in Bront et al. (2009) and presented above in Algorithm 1.

The function `determineOfferset`$(\boldsymbol{\pi}, \epsilon)$ calculates the set of products to offer depending on the current bid prices $\boldsymbol{\pi}$ via the greedy algorithm pointed out in Bront et al. (2009). As we put ourselves into a policy improvement setting and started out with the greedy strategy, the whole procedure tends to find solely other policies rather closeby, i. e. also greedy. Thus, we would leave out a big portion of the whole solution space (for each time period and each combination of capacities any subset of the n products can be offered). This is the

famous *exploration vs. exploitation dilemma*. Exploitation: The current solution was found and improved already and should be exploited even further (comparable to incremental innovation). Exploration: But also other, yet to be seen, parts of the solution space should be explored (comparable to disruptive innovation). To account for this dilemma, we follow Koch (2017) and use an *epsilon-greedy strategy*. With a probability of $\epsilon/2$ either no product is offered at all or all products with positive contribution $r_j - \sum_{h\in[m]} a_{hj} \cdot \pi_h$ are offered. With a probability of $1-\epsilon$, the set determined by Algorithm 1 is offered.

Note that in the very first iteration, we do not have appropriate values for $\boldsymbol{\pi}$ and θ. Thus, we decided to set them to zero at start, corresponding to "no opportunity costs of resources".

3.1.4 API update of parameters

With the information of the I sample paths, all optimization variables $(\Theta, \boldsymbol{\Pi})$ are updated. Note that the old parameters are used as starting values and the current policy iteration k is also passed to potentially take care of exponential smoothing. Thus, we have $(1+h)*T$ parameters (θ_t and $\boldsymbol{\pi}_t$ $\forall t \in [T]$) and TI data points, where each data point consists of the value of the value function as y-value and the current assignment of t and $\boldsymbol{\pi}_t$ as x-values.

The function `updateParameters`$(\hat{V}, \hat{\boldsymbol{C}}, \Theta, \boldsymbol{\Pi}, k)$ comprises the following least squares optimization problem, which optimizes all optimization variables, $(\theta_t, \boldsymbol{\pi}_t)_t$ with $t \in [T]$ at the same time and the optimal values $(\Theta^{update}, \boldsymbol{\Pi}^{update})$ are obtained.

$$\min \sum_{i=1}^{I} \sum_{t=1}^{T} \left(\hat{V}_t^i - V_{t\boldsymbol{c}_t^i}(\theta_t, \boldsymbol{\pi}_t) \right)^2 \tag{3.5}$$

$$\text{s.t.} \tag{3.6}$$

$$\theta_t \geq 0 \qquad \forall t \in [T] \tag{3.7}$$

$$\max_{j\in[n]} r_j \geq \pi_{th} \geq 0 \qquad \forall t \in [T], \forall h \in [m] \tag{3.8}$$

$$\theta_t \geq \theta_{t+1} \qquad \forall t \in \{1, \ldots, T-1\} \tag{3.9}$$

$$\pi_{th} \geq \pi_{t+1,h} \qquad \forall t \in \{1, \ldots, T-1\} \tag{3.10}$$

The old values θ_t^k and $\boldsymbol{\pi}_t^k$ are used to determine the optimal parameter θ_t^{update} and $\boldsymbol{\pi}_t^{update}$.

There exist multiple possibilities on which values shall be used as new optimization

variables. One approach might be to just use the optimal values:

$$\theta_t^{k+1} = \theta_t^{update} \tag{3.11}$$

$$\boldsymbol{\pi}_t^{k+1} = \boldsymbol{\pi}_t^{update} \tag{3.12}$$

Another approach is to use exponential smoothing, which weighs in the previous iterations and updates the optimization variables less and less, i. e. the weight for the updated optimization variable decreases with an increasing number of policy iterations k. Here, for the next iteration $k+1$ the values of optimization variables are calculated via:

$$\theta_t^{k+1} = \left(1 - \frac{1}{k}\right) \theta_t^k + \frac{1}{k} \theta_t^{update} \tag{3.13}$$

$$\boldsymbol{\pi}_t^{k+1} = \left(1 - \frac{1}{k}\right) \boldsymbol{\pi}_t^k + \frac{1}{k} \boldsymbol{\pi}_t^{update} \tag{3.14}$$

One can also apply the first technique (use the updates directly) during the policy iteration and finally report an average over all parameter values at the very end. Note that Lemma 11 presents the circumstances and mathematical proof on when this procedure becomes equivalent to exponential smoothing as introduced above.

$$\theta_t^{K+1} = \frac{1}{K} \sum_{k=1}^{K} \theta_t^k \tag{3.15}$$

$$\boldsymbol{\pi}_t^{K+1} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{\pi}_t^k \tag{3.16}$$

3.2 Methods using API as described by Koch (2017)

As we have outlined the general theory on Approximate Policy Iteration, we want to present several programs derived from this in the following. *API linear concave* is presented first, followed by *API piecewise linear concave* and finally *API piecewise linear* is described.

3.2.1 API linear concave

API linear concave is the one we used to introduce API. Its objective function is linear in the capacities and one of its constraints ensure concavity amongst the π's. The complete optimization problem shall be denoted by API-lc and is presented in the following.

$$\min \sum_{i=1}^{I} \sum_{t=1}^{T} \left(\hat{V}_t^i - V_{tc_t^i}(\theta_t, \boldsymbol{\pi}_t) \right)^2 \tag{3.17}$$

$$\text{s.t.} \tag{3.18}$$

$$\theta_t \geq 0 \qquad \forall t \in [T] \tag{3.19}$$

$$\max_{j \in [n]} r_j \geq \pi_{th} \geq 0 \qquad \forall t \in [T], \forall h \in [m] \tag{3.20}$$

$$\theta_t \geq \theta_{t+1} \qquad \forall t \in \{1, \ldots, T-1\} \tag{3.21}$$

$$\pi_{th} \geq \pi_{t+1,h} \qquad \forall t \in \{1, \ldots, T-1\} \tag{3.22}$$

with $V_{tc_t^i}(\theta_t, \boldsymbol{\pi}_t) \coloneqq \theta_t + \sum_{h \in [m]} \pi_{th} c_h$.

3.2.2 API piecewise linear concave

API-lc is just allowing for one parameter π_{th}, thus, it cannot distinguish between varying capacities of resource h during time period t. We want to introduce more flexibility into our model and thus split the initial available capacity c_h^0 for each resource up into S_h disjoint subsets given by the help of $S_h + 1$ thresholds $0 = b_h^0 < b_h^1 < \cdots < b_h^{S_h} = c_h^0$. Thus, each variable π_{th} is split up further into π_{ths} with $s \in \{1, \ldots, S_h\}$. Koch (2017) introduces a function $f_{hs}(c_h)$ to let π_{ths} be active (multiplied with value > 0) if the available capacity exceeds the threshold b_h^{s-1}. We want to opt for even greater flexibility and let just one of the π_{ths} be active for any specification of t and h. So, we introduce

$$f_{hs}(c_h) = \begin{cases} 0 & \text{if } c_h \leq b_h^{s-1} \\ c_h & \text{if } b_h^{s-1} < c_h \leq b_h^s \\ 0 & \text{if } b_h^s < c_h \end{cases} \tag{3.23}$$

The complete optimization problem is given by

$$\min \sum_{i=1}^{I} \sum_{t=1}^{T} \left(\hat{V}_t^i - V_t(\theta_t, \boldsymbol{\pi}_t, \boldsymbol{c}_t^i) \right)^2 \tag{3.24}$$

$$\text{s.t.} \tag{3.25}$$

$$\theta_t \geq 0 \qquad \forall t \tag{3.26}$$

$$\max_{j=1,\dots,n} r_j \geq \pi_{ths} \geq 0 \qquad \forall t, h, s \tag{3.27}$$

$$\pi_{ths} \geq \pi_{th,s+1} \qquad \forall t, h, s = 1, \dots, S_h - 1 \tag{3.28}$$

$$\theta_t \geq \theta_{t+1} \qquad \forall t = 1, \dots, T-1 \tag{3.29}$$

$$\pi_{ths} \geq \pi_{t+1,hs} \qquad \forall t = 1, \dots, T-1 \tag{3.30}$$

with $V_t(\theta_t, \boldsymbol{\pi}_t, \boldsymbol{c}_t) \coloneqq \theta_t + \sum_{h=1}^{m} \sum_{s=1}^{S_h} \pi_{ths} f_{hs}(c_h)$ and $f_{hs}(c_h)$ as specified in Equation (3.23).

3.2.3 API piecewise linear

This problem is almost as above, but removes the concavity constraint on the π_{ths}. Its optimization problem is described by

$$\min \sum_{i=1}^{I} \sum_{t=1}^{T} \left(\hat{V}_t^i - V_t(\theta_t, \boldsymbol{\pi}_t, \boldsymbol{c}_t^i) \right)^2 \tag{3.31}$$

$$\text{s.t.} \tag{3.32}$$

$$\theta_t \geq 0 \qquad \forall t \tag{3.33}$$

$$\max_{j=1,\dots,n} r_j \geq \pi_{ths} \geq 0 \qquad \forall t, h, s \tag{3.34}$$

$$\theta_t \geq \theta_{t+1} \qquad \forall t = 1, \dots, T-1 \tag{3.35}$$

$$\pi_{ths} \geq \pi_{t+1,hs} \qquad \forall t = 1, \dots, T-1 \tag{3.36}$$

with $V_t(\theta_t, \boldsymbol{\pi}_t, \boldsymbol{c}_t) \coloneqq \theta_t + \sum_{h=1}^{m} \sum_{s=1}^{S_h} \pi_{ths} f_{hs}(c_h)$ and $f_{hs}(c_h)$ as specified in Equation (3.23).

3.3 Application to working example

Now, we want to exemplarly apply API-lc to our working example to arrive at the final values for the parameters θ_t and $\pi_{t,h}$ that can be used later to compare it with various

other methods for solving capacity control. We first outline the concrete setup of the policy iteration, then elaborate on how sample paths are generated, and finally analyse the training process.

We evaluate different combinations of parameters and go with the same as Koch (2017), i. e. $K = 60$ policy iterations, $I = 800$ sample paths and a parameter for the epsilon-greedy strategy monotonously decreasing with the number of policy iteration k:

$$\epsilon_k = \begin{cases} 0.05, & k = 1, \dots, 10 \\ 0.01, & k = 11, \dots, 20 \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

We prepare $I \times T$ random numbers for the customer streams, the sales stream and the epsilon criterion. Note that these numbers are random, but remain the same over the K policy iterations. Figure 3.2 shows the distribution of a uniform random variable in the whole $I \times T$ setting. It can be clearly seen that the underlying random variable indeed follows a uniform $U(0, 1)$ distribution and can thus be used for applying the epsilon-greedy strategy with thresholds as specified above. Figure 3.3 presents the boxplots of arriving customers. For each of the I iterations, the number of arriving customers in the event horizon ($t \in [T]$) was reported and then depicted. The means can be seen to be in line with the model specification (e. g. customer 1 has arrival probability of 0.3) and the standard deviation is fine. Thus, our exemplary data represents our model specification and we can apply approximate dynamic programming.

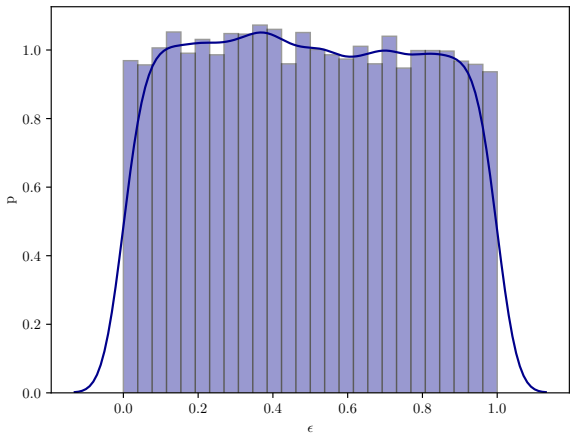


Figure 3.2: Distribution of epsilon values.

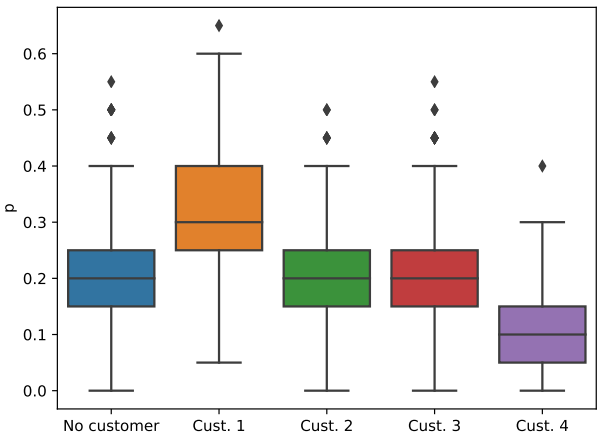


Figure 3.3: Boxplot of customers in training.

The evolution of the average realized value[4] over the I iterations in each of the $k \in \{1, \ldots, 60\}$ policy iterations is depicted in Figure 3.4. It can be seen that the initial simplification of having opportunity costs of zero ($k = 1$) results in a suboptimal result. The average value increases after 10 policy iterations and reaches it optimum after 20 policy iterations. As these jumps coincide with the change of the ϵ parameter for the epsilon-greedy strategy, it seems as if this strategy is not worth applying in this particular setting (our working example).

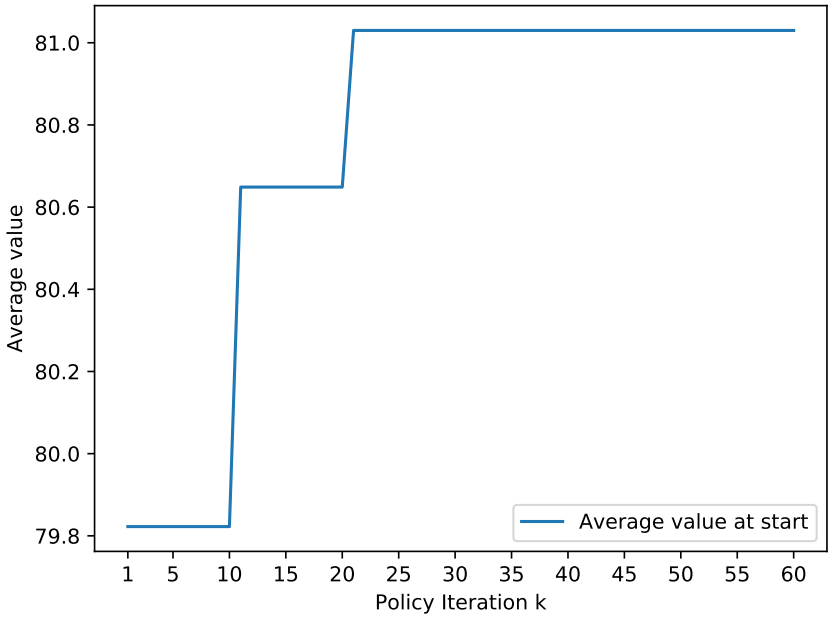


Figure 3.4: Average value at start (y-axis) for the evolution via $K = 60$ policy iterations (x-axis).

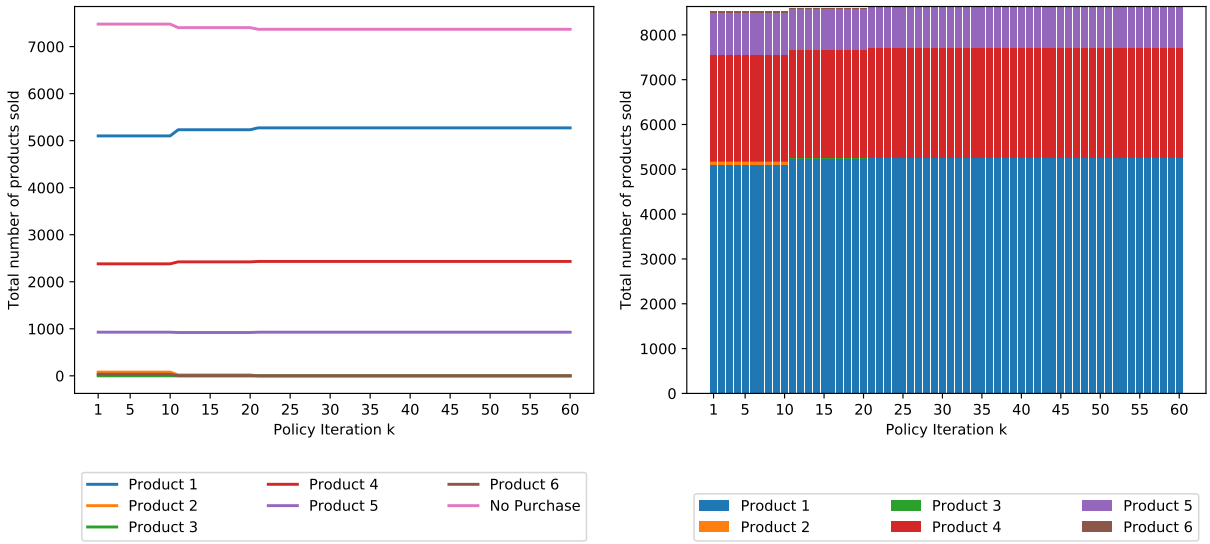
That some resources are still left over after the time horizon can be seen in Table 3.2. Again, slight changes can be observed after 10 and after 20 iterations. Resources 1 and 3 are used more and more (less remaining capacity), while Resource 2 is first used less and then hold constant. Resource 4 is first used less and then more.

[4]Realized value refers to value realized over the time horizon, i. e. total revenue generated due to selling of products in time periods $t \in [T]$.

	Resource 1	Resource 2	Resource 3	Resource 4
1-10	1.48	3.96	1.02	6.84
11-20	1.43	4.00	0.97	6.85
21-60	1.41	4.00	0.96	6.84

Table 3.2: Average remaining capacities via $K = 60$ policy iterations in the working example.

These slight changes can also be seen by looking at Figure 3.5, which depicts the purchased products. Figure 3.5a includes the "no-purchases" explicitly, while Figure 3.5b only depicts purchased products stacked over each other, leading the total height of each cumulated bar to represent the total amount of purchased products. Note that summing up all purchase actions (including "no-purchase"), leads to a total of $16,000 (= 20 \cdot 800 = T \cdot I)$ in each polity iteration k. Again, the pattern of the value plot (Figure 3.4) and of the remaining capacities (Table 3.2) can be identified. There are slight changes after 10 and after 20 iterations, but almost no changes between policy iterations. Product 1 is purchased more and more over time (from $5,101$ over $5,232$ to 5272 times), while the cheap product 2 and the medium priced product 6 are purchased less and less. Product 3 is never purchased at all (what does not surprise as no customer segment considers this product). The number of no-purchases decreases slightly (from $7,478$ over $7,403$ to $7,368$).



(a) With no-purchases.

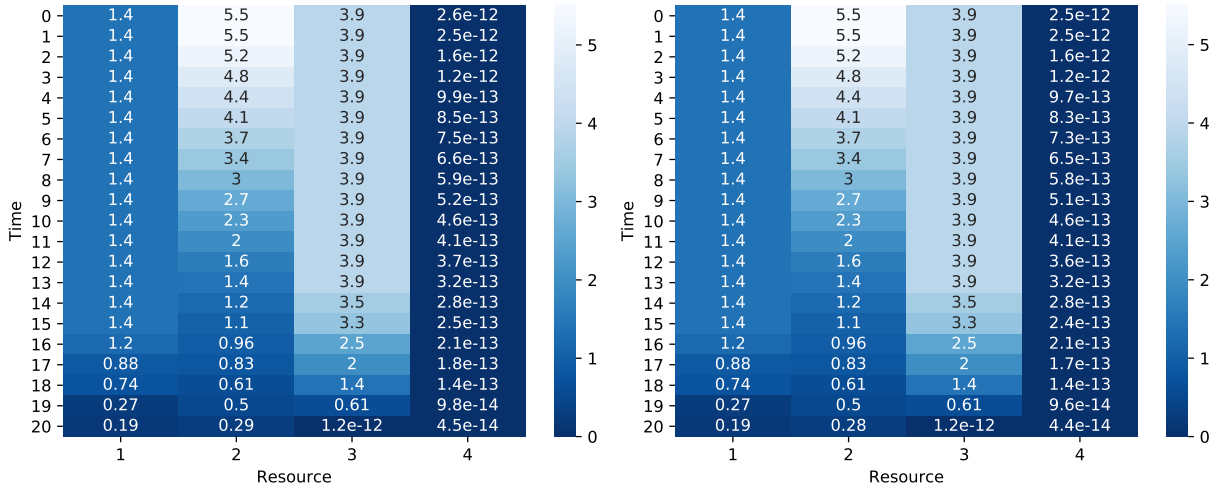
(b) Solely products.

Figure 3.5: Evolution of purchased products (y-axis) via $K = 60$ iterations (x-axis).

	[5]	[4]	[3,5]	[3,4,5]	[1]	[1,3,4,5]	[1,2,3,5,6]	[1,2,3,4,5,6]	[]
1-10	153	797	6	24	1783	12506	38	297	396
11-20	187	853	0	7	1965	12834	8	56	90
21-60	192	871	0	0	1989	12948	0	0	0

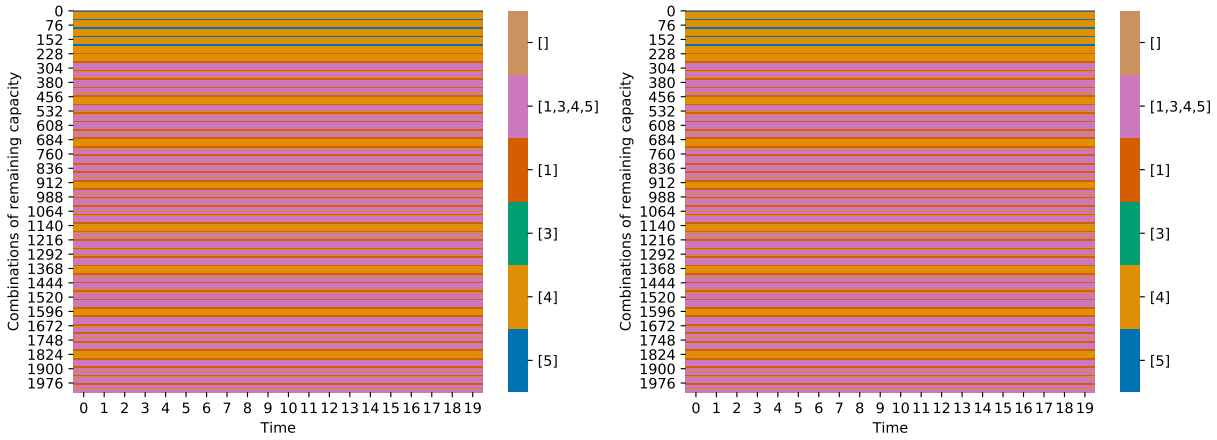
Table 3.3: Sets offered via $K = 60$ policy iterations in the working example.

Let us now explore the resulting optimal values and offer sets using two more graphs. Figure 3.6 presents the optimal values of π_{th} for every time period t and resource h. Note that the optimal values do barely change from the second-last iteration (Figure 3.6a) to the last iteration (Figure 3.6b). Indeed, changes occur at the forth decimal place. These slight changes do not affect the finally offered set as can be seen in Figure 3.7. This figure presents the optimal offersets to be offered at any time (x-axis) and for any combination of resource capacities (y-axis) in a separate colour. By "combination of resource capacities", we refer to the following: As the vector of starting capacities for resources is given by $(8, 4, 4, 8)$ and the least amount of capacity is zero, there are a total of $9 \cdot 5 \cdot 5 \cdot 9 = 2025$ combinations possible. Note however, API with linear value function approximation abstracts from these combinations, as it is not aware of the amount of available capacity. We see that for most states, the set $\{1, 3, 4, 5\}$ is offered. If that is not possible (due to not all necessary resources available), API offers just a single product, or none at all (but this just if there is no capacity at all).



(a) $k = 59$ (b) $k = 60$

Figure 3.6: Heatmap of optimal values for π_{th} for the two final policy iterations.



(a) $k = 59$ (b) $k = 60$

Figure 3.7: Categorical map of offersets for all combinations of remaining capacity for the two final policy iterations.

Chapter 4

Neural networks

We now want to build upon Approximate Policy Iteration and combine it with a new method, Neural Networks. Neural Networks, and their subcategory Deep Neural Networks, are major topics in Artificial Intelligence and also have major public attention currently. In the following, we want to pave a path to apply Neural Networks in our setting of revenue management. Section 4.1 introduces linear regression, which will be used as a baseline. Section 4.2 introduces theory on neural networks, while Section 4.4 focuses on how hyperparameters (parameters to characterize the neural network) can be trained. Finally, neural networks are applied to our working example.

All the following procedures are special instances of API. We have already introduced different versions of AI by changing the value function (e. g. API-lc vs. API-plc) or by considering different constraints (e. g. API-plc vs. API-pl). The following procedures will introduce other approaches for approximating the value function, don't have constraints on their parameters and might alter the way how offersets are determined. The general theory and its notation are oriented at well established textbooks, e. g. Murphy (2012) and Bishop (2009).

4.1 Linear regression

Linear regression can be seen as a subclass of Neural Networks.[1] But as linear regression is closely related to Neural Networks, as the reader is potentially already familiar with it[2] and as another special version of linear regression was already used in Equation (3.2), we

[1] A neural network with the identity as activation function is a linear regression model.

[2] Regression is taught in many undergraduate courses nowadays. Note that the term "regression" was coined by Galton (1886). He described the phenomena that the body height of kids was not inherited from their parents, but that the body height of kids trended towards the average body height.

want to introduce it again formally and present methods to solve it. Note that we apply basic theory of multivariate calculus here.[3] We then apply it directly to our setting of revenue management, before bridging the gap to neural networks.

Linear regression aims at establishing a (linear) connection between some D-dimensional feature data $\boldsymbol{x} \in \mathbb{R}^D$ and its associated value $y \in \mathbb{R}$. Usually a whole set of data samples is considered, each individual j with features in accordance with $\boldsymbol{x}_j$ (row vector) and value y_i and all together characterized by the so called *design matrix* $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and vector of *training labels* $\boldsymbol{y} \in \mathbb{R}^N$. Linear regression asserts that the response is a linear function of its inputs, i. e.

$$y \approx f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} + \epsilon = \sum_{j=1}^{D} w_j x_j + \epsilon$$

where $\boldsymbol{w}^\intercal \boldsymbol{x}$ represents the scalar product between the input vector $\boldsymbol{x}$ and the *weight vector* $\boldsymbol{w}$, and ϵ is the *residual error* between the linear predictions and the true response.

Metrics for the goodness of a linear regression $f_{\boldsymbol{w}}$ are called *error* or *loss function* $E(\cdot)$. Widely used is *least squares*, which builds on the sum of squared errors (SSE):

$$E_{LS}(\boldsymbol{w}) = \frac{1}{2} \sum_{j=1}^{N} \left(f_{\boldsymbol{w}}(\boldsymbol{x}_j) - y_j \right)^2 \tag{4.1}$$

$$= \frac{1}{2} \sum_{j=1}^{N} \left(\boldsymbol{w}^\intercal \boldsymbol{x}_j - y_j \right)^2 \tag{4.2}$$

The optimal weight vector $\boldsymbol{w}^*$ is then given by

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E_{LS}(\boldsymbol{w}) \tag{4.3}$$

$$= \arg\min_{\boldsymbol{w}} \frac{1}{2} \sum_{j=1}^{N} \left(\boldsymbol{w}^\intercal \boldsymbol{x}_j - y_j \right)^2 \tag{4.4}$$

$$= \arg\min_{\boldsymbol{w}} \frac{1}{2} \left(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \right)^\intercal \left(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \right) \tag{4.5}$$

The minimum of $E_{LS}(\boldsymbol{w})$ can now be found as usual by computing the first derivative,

[3] As a refresher, one might take a look at Courant et al. (2000).

called *gradient* in the multidimensional case, $\nabla_{\boldsymbol{w}} E_{LS}(\boldsymbol{w})$:

$$\nabla_{\boldsymbol{w}} E_{LS}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\intercal (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \tag{4.6}$$

$$= \nabla_{\boldsymbol{w}} \frac{1}{2} (\boldsymbol{w}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{w}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{y}^\intercal \boldsymbol{y}) \tag{4.7}$$

$$= \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^\intercal \boldsymbol{y} \tag{4.8}$$

Now this gradient has to be set to zero and solved for *boldsymbolw* to obtain the minimizer[4]. This leads to the *normal equation* in Equation (4.10) of the least squares problem with $\boldsymbol{X}^+$ denoting the Moore-Penrose pseudo-inverse.

$$\boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^\intercal \boldsymbol{y} \stackrel{!}{=} 0 \tag{4.9}$$

$$\boldsymbol{w}^* = \underbrace{(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal}_{=\boldsymbol{X}^+} \boldsymbol{y} \tag{4.10}$$

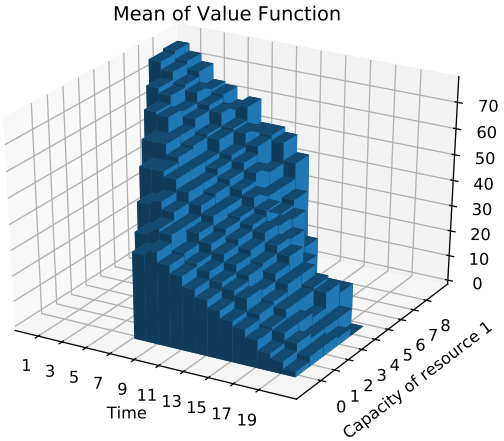
In our setting, an individual feature data point $\boldsymbol{x}_j$ is given by the current time t and capacity $\boldsymbol{c}$, i. e. $\boldsymbol{x}_j = (t, c_1, \ldots, c_m)$. The linear regression is implemented in Python with `statsmodels.api.OLS`, which indeed uses the outlined closed-form solution.[5]

Now it comes to choosing which variables we want to include in our model. We start by taking a look at the data to let us be guided through the process of choosing the variables. Figure 4.1 depicts the mean value of all samples going through the specified time and capacity of the different resources. This means that per cell, time t and capacity of one of the resources c_h is fixed and the other capacities can take all values. Interestingly, we see that sometimes, the mean value of a state with less capacity is greater than for a cell with higher capacity, even if time is fixed. This can be seen e. g. for resource 3 in Figure 4.1c for time 1 and capacity 3 vs capacity 4. This might seem counter-intuitive, that is why we also want to consider a short example to highlight the identified problem.

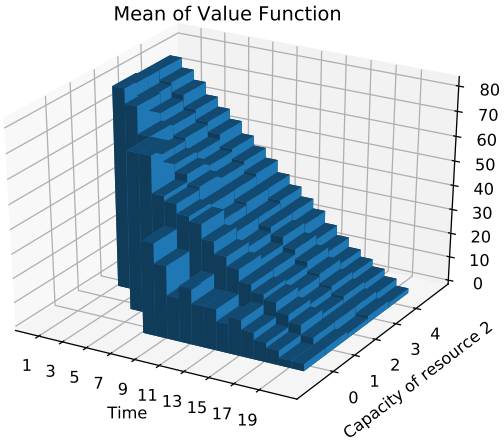
Example 1. (Mean value for less capacity is greater than for more capacity) Consider two sample paths on a tiny example with an event horizon of $T = 2$, two products with revenues $r = (10, 100)$ and one resource with starting capacity $c^0 = 2$. Their expected values for

[4]Note that this is indeed the minimizer as the "second derivative", which is called *Hessian* in the multivariate case, $\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} E_{LS}(\boldsymbol{w})$ is positive (semi)definite.

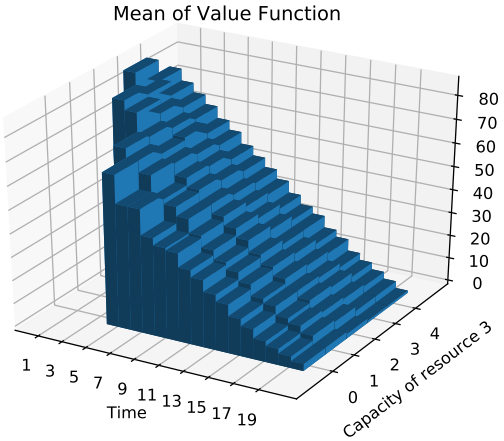
[5]At this point, we want to point out that another major machine learning package `sklearn.linear_model.LinearRegression` used to use the numerical method of Steepest Gradient Descent before also switching to OLS as pointed out on Stackoverflow.



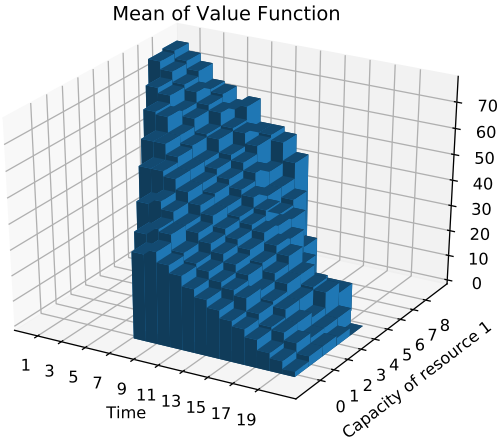
(a) Varying capacity on leg 1.



(b) Varying capacity on leg 2.



(c) Varying capacity on leg 3.



(d) Varying capacity on leg 4.

Figure 4.1: Mean value of working example with time $t \in [20]$ on x-axis, capacity as stated on the y-axis (average over all walk-throughs, no matter what the other resources had for capacity[6]) and the mean value depicted on z-axis.

each state (c, t)[7] are given in the first two tabulars of Table 4.1. A star * indicates a sales event happening in this state (leading to immediate revenue). The third table depicts the mean values generated when considering for each cell just the paths that were indeed taken. This means that the mean value for $mv(c = 2, t = 0) = \frac{20+200}{2}$, but $mv(c = 2, t = 1) = \frac{20}{1}$. Thus, for time $t = 1$, less capacity goes along with a higher mean value. This is highlighted by the boxes in the table. As one can easily show, this does not change when one considers unvisited states with a value of 0 (or any other constant).

[7] We switch notation as it seems to be more intuitive to have time in the columns, meaning time passes when moving to the right. Then standard matrix notation lets time t be in the second place.

Sample path 1		$t=0$	$t=1$	$t=2$
	$c=2$	20	20*	
	$c=1$			10*
	$c=0$			
Sample path 2		$t=0$	$t=1$	$t=2$
	$c=2$	200*		
	$c=1$		100*	
	$c=0$			0
Mean Values		$t=0$	$t=1$	$t=2$
	$c=2$	110	$\boxed{20}$	
	$c=1$		$\boxed{100}$	10
	$c=0$			0

Table 4.1: Example to illustrate mean value for less capacity is greater than for more capacity

Both observations (empirically in the figure and theoretically in the example) and the nonlinearity in time and capacity lead us to introduce separate variables for each state. That is, we have a total of T variables for time, denoted by $\Theta = (\theta_t)$, $t \in [T]$, and a total of $T \cdot m$ resource variables, denoted by $\mathbf{\Pi} = (\pi_{th})$, $t \in [T]$, $h \in [m]$. Note the similarity to linear API.

We now summarize the application of linear regression in our revenue management setting by the means of policy iteration. Figure 3.1 applies as well. That is, we first initialize all θ_t and π_{th} by 0 to generate the first $I = 800$ sample paths with revenues given by $\hat{V}_t^i$ and capacities $\boldsymbol{c}_t^i$. Then, we apply the epsilon-greedy strategy as described in Section 3.1.3. The function `updateParameters`$(\hat{V}, \hat{\boldsymbol{C}}, \Theta, \mathbf{\Pi}, k)$ is changed to return the optimal weight determined by ordinary least squares, i. e.

$$\underset{(\Theta,\mathbf{\Pi})}{\arg\min} \sum_{i=1}^{I} \sum_{t=1}^{T} \left(\hat{V}_t^i - (\theta_t \cdot 1 + \boldsymbol{\pi}_t^\intercal \hat{\boldsymbol{c}}_{\boldsymbol{t}}^{\boldsymbol{i}}) \right)^2$$

Exponential smoothing is applied to attain the new parameters leading to the new offersets for the next $I = 800$ sample paths. This procedure is repeated for $K = 60$ iterations.

4.2 General theory on neural networks

We now move on to Neural Networks and how these can be applied to our setting. Before we do so, we want to give a brief overview on the history of Neural Networks, then explain the set up of a neural network and how it is trained in general, and finally elaborate on how we apply it in our setting of revenue management. Again, notation is vastly in line with the well established textbooks Murphy (2012) and Bishop (2009).

The term "neural network" originated in attempts to mathematically represent the information processing of biological systems and was started half a century ago by McCulloch and Pitts (1943), Widrow and Hoff (1960) and Rosenblatt (1961) amongst others. It tries to benefit from the structure and functioning of the human brain and has been used broadly and with success in the realms of biology. But as computational power was rather limited in the 1960's and less data were available, the application domain was restricted. Nowadays, computational power is increasing exponentially[8] and data is collected and available in the age of Big Data. This has lead to increased interest in artificial intelligence and to many different research disciplines trying to benefit from its methods. Accordingly, also the German government has decided on a strategy of Artificial Intelligence in November of 2018.[9]

We now want to focus on the *Multilayer Perceptron* (MLP), a specific class of neural networks that according to Bishop (2009) "have proven to be of greatest practical value". The goal of MLP is to either solve a classification or a regression problem. In general, the network consists of several layers: one *input layer*, potentially multiple[10] *hidden layers* and one *output layer*. Each layer contains multiple nodes, which are called *neurons*. Neurons of different, consecutive layers might be connected by edges, but there are no connections amongst the neurons of one layer. Each neuron represents the basic component of a MLP and its functionality resembles a biological neuron. Input signals are received, processed and output signals are generated. The whole network itself takes inputs, processes these via multiple layers and reports an output. This process of feeding information from one layer to the next is referred to as *Feed Forward*. Let us consider the network depicted in

[8] Note the famous *Moore's Law* by Gordon Moore. He said in 1965 that the number of transistors that would fit in a given circuit space was doubling every year. This has been true untrue until some years ago, but now computer hardware is reaching physical boundaries as manufacturing technology is now able to place electrical wires at a distance of 7 nanometers and quantum problems like electron tunneling start to occur.

[9] https://www.bundesregierung.de/resource/blob/975226/1550276/3f7d3c41c6e05695741273e78b8039f2/2018-11-15-ki-strategie-data.pdf?download=1

[10] A Neural Network with more then one hidden layer is referred to as a Deep Neural Network (DNN).

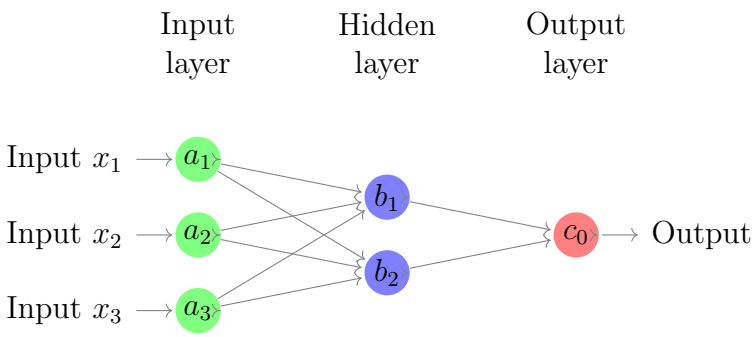


Figure 4.2: Neural Network explanatory example.

Figure 4.2 as an example for going over its components and for explaining its functionality. The most important notation is presented in Table 4.2 for a reference.

Symbol	Domain	Meaning
$i_n^{(i)}$	$\mathbb{R}$	input variable for layer i, node n
$o_n^{(i)}$	$\mathbb{R}$	output variable for layer i, node n
$w_{jk}^{(i)}$	$\mathbb{R}$	weight for layer $i \geq 2$, to be multiplied with $o_k^{(i-1)}$

Table 4.2: NN overview of parameters.

Figure 4.2 presents a network with 3 layers. The input layer receives three inputs, x_1, x_2, x_3, which are taken into the nodes a_1, a_2, a_3 as inputs. We want to denote the input of the first layer for the n-th node by $i_n^{(1)}$.There is not much happening in the input layer, as it just takes the input variables and passes them on as output. One could say that to get the output, one applies the identity function on the input. We denote the output of the first layer for the n-th node by $o_n^{(1)}$. To sum it up, we look at the output generated by the first layer, n-th node:

$$o_n^{(1)} = id(i_n^{(1)}) = id(x_n) = x_n$$

Now we move from the input layer to the hidden layer. As our MLP is fully connected, the output values of all the three input nodes are passed on to all nodes of the hidden layer. At this layer, something is happening with its input variables. We look at the n-th node of the hidden layer b_n. First, all its inputs are linearly combined with associated weights and a bias term is added to derive the input value to the second layer, n-th node (this

calculation is sometimes referred to as *propagation*)

$$i_n^{(2)} = \sum_{k=1}^{3} w_{n,k}^{(2)} \cdot o_k^{(1)} + w_{n,0}^{(2)}$$

where the 3 belongs to the three nodes of the input layer, and the $^{(2)}$ indicates that the weights are associated with the second layer (hidden layer). These quantities $i_n^{(2)}$ are referred to as *activations*. Each of them is then transformed using a differentiable, nonlinear *activation function* $f(\cdot)$ to produce the output of the second layer, n-th node

$$o_n^{(2)} = f\left(i_n^{(2)}\right)$$

This procedure could now be repeated for any number of hidden layers, with the output layer being the last. In our example, we have already arrived at the output layer with just one node c_1 and can calculate the value of third layer, first node

$$o_1^{(3)} = f(i_1^{(3)}) = f\left(\sum_{k=1}^{2} w_{n,k}^{(3)} \cdot o_{2,k} + w_{n,0}^{(3)}\right)$$

Now we already understood the structure of a network, but two questions remain. First, which activation functions are commonly used? And second, how are the weights determined? We explore this in the following.

4.2.1 Activation Functions

In general, any differentiable, nonlinear function $f(\cdot) : \mathbb{R} \to \mathbb{R}$ can be used as activation function. But the following functions are most regularly used and widely implemented. We also elaborate a bit on the structure of the function and eventually comment on its application in machine learning.

Definition 2. (Identity) The *identity* function $f : \mathbb{R} \to \mathbb{R}$ is given by

$$f(x) = x \ .$$

If the identity is applied as activation function, the neural network basically performs linear regression.

Definition 3. (Logistic) The *logistic* function $f : \mathbb{R} \to [0, 1]$ is given by

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \ .$$

It takes any number on the real number line and squeezes it into the interval $[0, 1]$. Very small values are taken close to 0, very large values taken close to 1. Furthermore, it is strictly monotonously increasing. As such, it resembles an S-shape.

This logistic function is often used for neural networks being trained by a back-propagation algorithm, as the derivative is easy to compute as can be seen in Karlik and Olgac (2011).

Definition 4. (Hyperbolic Tangens) The *tangens hyperbolicus* $f : \mathbb{R} \to [-1, 1]$ is given by

$$f(x) = \tanh(x) = \frac{\mathrm{e}^{2x} - 1}{\mathrm{e}^{2x} - 1} \ .$$

It takes any number on the real number line and squeezes it into the interval $[-1, 1]$. Very small values are taken close to -1, very large values taken close to 1. Furthermore, it is strictly monotonously increasing and symmetric to the origin. As such, it also resembles an S-shape (as the logistic function).

Definition 5. (Rectified linear unit) The *rectified linear unit* $f : \mathbb{R} \to \mathbb{R}_0^+$ is given by

$$f(x) = \mathrm{ReLU}(x) = \max(\{0, x\}) \ .$$

It takes any number and sets it to zero if it was negative. Thus, it is continuous, but not differentiable at $x = 0$.

Note that the rectified linear unit is not differentiable, which prohibits it to be used in the context of neural networks (theoretically). But it actually works fine in practice and is computationally fast, so that is widely used.

4.2.2 Training a Neural Network

Now we are ready to explore how a neural network is trained. This process is also called "fitting" or "learning". The overall goal is to model the relationship between D-dimensional feature data $\boldsymbol{x} \in \mathbb{R}^D$ and its associated values $y \in \mathbb{R}$ via a neural network. So one can also see the neural network as a function $n_{\boldsymbol{W}}(\cdot)$ taking a feature vector $\boldsymbol{x}$ and mapping it

to its predicted output by the usage of the various layers with its associated weights and activation functions. All the weights of the different layers are denoted by $\boldsymbol{W}$. Training now refers to using N sample data points to derive the optimal values for all the weights $w_{jk}^{(i)}$. And the process also uses an error function, just like linear regression does as described in Section 4.1. Again, we apply least squares as error function E_{LS}[11]

$$E_{LS}(\boldsymbol{W}) = \frac{1}{2}\sum_{j=1}^{N}(n_{\boldsymbol{W}}(\boldsymbol{x}_j) - y_j)^2$$

Now, errors are computed and ideas like gradient descent (as outlined in Section 4.1) are used to optimize the weights. In practice, stochastic gradient descent can be used. The Adam algorithm is based on stochastic gradient descent and often used in practice (also by us). Adam is an algorithm for first-order gradient-based optimization and is based on adaptive estimates of lower-order moments. More details can be found in Diederik P. Kingma and Jimmy Ba (2014). Another popular algorithm to be used in training is Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), which belongs to the family of quasi-Newton methods with more details in Malouf (2002).

The actual process of adapting the weights is done iteratively over the different layers, starting from the last layer. The process itself is referred to as *backpropagation* and is outlined in the following in more depth in e. g. Bishop (2009).

4.3 Applying a neural network in revenue management

Above, we used linear regression to approximate the value function, derive updated parameters θ_t and π_{th} and apply those in `determineOfferset` to determine the set of products to be offered. Our idea of using neural networks is to skip the intermediate step of using parameters θ and π and to use the value function approximation via a neural network directly. The set of products to be offered is then calculated via a combination of Bellman's equation and the greedy heuristic described in Algorithm 1. This requires an in depth discussion.

We embed the neural network in an approximate policy iteration setting. That means, at the first policy iteration, for each sample path, in every state, all possible products[12]

[11] Note that the error function can easily be changed to also be capable of handling multidmensional $\boldsymbol{y}$. The trick is to use a norm $\|n_{\boldsymbol{W}}(\boldsymbol{x}_j) - \boldsymbol{y}\|$, where we might take the square norm $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\intercal \boldsymbol{x}}$.

[12] A product is possible if enough capacity of its needed resources are available.

are offered. This leads to the first set of training data with states given by $(t^i, \boldsymbol{c}_{t^i})$ and corresponding value y_{t^i}, i. e. a total of $T \cdot I$ data points. These data points are now taken as input to train the first network.

Let us now take a look at the structure of our model. We first need to specify the input layer. To be as flexible as possible, we split the variables of one particular state $(t, c_1, \ldots, c_m)$ up similar to the one-hot vector notation. The one dimensional variable t is mapped to T nodes of the input layer, passing a value of 1.0 to node t and 0.0 to all others. In our notation from above, we can say that the input value of the first layer, n-th node is given by $i_n^{(1)} = \mathbb{1}_{\{n=t\}}$ for $n \in [T]$, i. e. just for the nodes belonging to the time. We proceed in a similar way for the capacities. Also each resource h is represented by a total of T input nodes with just the current (t) one having a value different to zero. But this time, the value is not 1.0, but the actual capacity c_h^t. In other words, when considering the n input nodes that take care of resource h, their corresponding input values are given by $i_n^{(1)} = c_h \cdot \mathbb{1}_{\{n=t\}}$ for $n \in [T]$.

To sum it up: The input layer of our neural network consists of $T \cdot m$ nodes, with just $1 + m$ of them being "active" (having a value different to 0) for any given data point.

The hidden layer structure will be given by the result of our hyperparameter tuning and we have one output node that shall estimate the value y for the particular state $(t, c_1, \ldots, c_m)$. The activation function will also be determined by hyperparameter optimization.

With the given sample data at hand, we train our neural network NN and return the complete NN (instead of parameters θ_t and π_{th}, when comparing to API). The first policy iteration is completed.

During the second policy iteration, the neural network NN is used for each sample path, for each state $(t, c_1, \ldots, c_m)$ to determine the offersets in `determineOfferset`. For this, we also use an adapted version of the Greedy Heuristic as presented in Algorithm 2. Essentially, we replaced $A_i^\intercal \pi$ with the function $\texttt{oppCost}(j, t, \boldsymbol{c}, NN)$, which is calculated by

$$\texttt{oppCost}(j, t, \boldsymbol{c}, NN) = \begin{cases} \infty & \text{if not enough resources to offer product } j \\ NN((t+1, \boldsymbol{c})) - NN((t+1, \boldsymbol{c} - \boldsymbol{a}_j)) & \text{otherwise} \end{cases}$$

With this method for determining the offersets, the sample data is generated for the second policy iteration. The complete neural network (fixed structure) is updated and used for the next policy iteration to determine offer sets. This process is continued for all

policy iterations.

1: $\texttt{Value}(X) \coloneqq \sum_{l=1}^{L} \lambda_l \frac{\sum_{i \in X}(r_i - \texttt{oppCost}(j,t,\boldsymbol{c},NN))u_{li}}{\sum_{i \in X} u_{li} + u_{l0}}$
2: $S \coloneqq \emptyset, \quad S' \coloneqq \{j \in [n] : r_j - \texttt{oppCost}(j,t,\boldsymbol{c},NN) > 0\}$
3: **if** $S' = \emptyset$ **then return** $\emptyset$
4: $j^* \coloneqq \arg\max_{j \in S'} \texttt{Value}(\{j\})$
5: **repeat**
6: $\quad S \coloneqq S \cup \{j^*\}, \quad S' \coloneqq S' \backslash \{j^*\}$
7: $\quad j^* \coloneqq \arg\max_{j \in S'} \texttt{Value}(S \cup \{j\})$
8: **until** $\texttt{Value}(S \cup \{j^*\}) \leq \texttt{Value}(S)$
9: **return** S, e. g. as offerset $\boldsymbol{x}$ given by $x_j = \mathbb{1}_{\{j \in S\}}, \ j \in [n]$

Algorithm 2: Greedy Heuristic adapted to work with a Neural Network.

4.4 Neural networks hyperparameter tuning

As we use `sklearn.neural_network.MLPRegressor` to implement the neural network, we have a few hyperparameters that can be optimized. We use a simple grid search for choosing the best combination of activation function, parameter α and the hidden layer size. In the following, we explain the process of our grid search.

For each hyperparameter, possible values were pre-specified in advance. Sample data has been created in the first policy iteration, when all possible products (with enough capacity of needed resources) are offered. Then, for each combination of hyperparameters, a neural network is trained and evaluated on several metrics.

Before moving on, the metrics for evaluating a neural network have to be introduced. For this, let y denote the correct value of the target variable and $\hat{y}$ the estimated target variable. Their components are indexed by i and we consider a total sample size of n_{samples}.

Definition 6. (r2) The *r2 score* computes the coefficient of determination, usually denoted as R^2, and is calculated via

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \ ,$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$.

The best possible score is 1.0 and negative scores are possible (because the model can be arbitrarily bad). A constant model that always predicts the expected value of y, no matter the input values, would get $R^2 = 0.0$.

Note that R^2 is a very popular measure all over statistics and represents the proportion of variance (of y) explained by the independent variables ($\boldsymbol{x}$) in the model. We want to point out that all our neural networks have relative low R^2. But we still apply the neural network and will see that it performs quite well, i. e. generates good values.

Definition 7. (explvar) The *explained variance regression score* is estimated as follows

$$explained\_variance(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

The best possible score is 1.0, lower values are worse.

Definition 8. (mae) The *mean absolute error* computes the mean of all absolute errors and is estimated via

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| .$$

The best possible value is 0.0, higher values are worse.

This risk metric corresponds to the expected value of the absolute error or $l1$-norm loss.

Definition 9. (msq) The *mean squared error* computes the mean of all squared errors and thus is calculated by

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

The best possible value is 0.0, higher values are worse.

This risk metric corresponds to the expected value of squared loss.

Definition 10. (msqlog) The *mean squared logarithmic error* computes the mean of all squared logarithmic errors and thus is calculated by

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \left(\log_e(1 + y_i) - \log_e(1 + \hat{y}_i)\right)^2 ,$$

where $\log_e$ represents the natural logarithm of x.

The best possible value is 0.0, higher values are worse.

This metric is best used when targets have exponential growth, such as population counts (not applicable in our setting).

Applying these metrics to our scenario leads to results as depicted in Table 4.3. We thus take as hyperparameters: tanh as activation function, $\alpha = 0.1$, and two hidden layers, one with 50 nodes and the second one with 10 nodes.

We just used a simple grid search for hyperparameter tuning. In future research however, more advanced techniques should be used to fine tune the model even further. One might consider using Bayesian optimization for hyperparameter tuning as described in Jasper Snoek et al. (2012) or in Bergstra et al. (2013).

4.5 Application to working example

The following analysis is done analogously to Section 3.3 in order to have a consistent evaluation of newly introduced models. Most of the found results for API-lc will also be observed in a similar way here for NN, but the specific values are different.

Again, we are using the same parameters for the overall API as Koch (2017), i. e. $K = 60$ policy iterations, $I = 800$ sample paths by the usage of the same $I \times T$ random numbers for the customer streams, the sales stream and the epsilon criterion. The used parameter for the epsilon-greedy strategy is again monotonously decreasing with the number of policy iteration k:

$$\epsilon_k = \begin{cases} 0.05, & k = 1, \dots, 10 \\ 0.01, & k = 11, \dots, 20 \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

The evolution of the average realized value[13] over the I iterations in each of the $k \in \{1, \dots, 60\}$ policy iterations is depicted in Figure 4.3. The initial offering of all possible products is improved immediately and more improvements occur at the steps of the epsilon parameter. Again, it seems like applying the epsilon-greedy strategy is not beneficial in our particular setting (the working example).

[13] Realized value refers to value realized over the time horizon, i. e. total revenue generated due to selling of products in time periods $t \in [T]$.

	activation	alpha	hidden_layer_sizes	r2	mae	explvar	msq	msqlog
0	identity	0.001	(10,)	0.0016	24.3284	0.0016	840.8795	0.9957
1	identity	0.001	(50, 10)	0.0011	24.3038	0.0020	841.3509	0.9848
2	identity	0.001	(100, 10, 10)	0.0005	24.3184	0.0012	841.8778	0.9857
3	identity	0.010	(10,)	0.0016	24.3284	0.0016	840.8807	0.9957
4	identity	0.010	(50, 10)	0.0011	24.3038	0.0020	841.3513	0.9848
5	identity	0.010	(100, 10, 10)	0.0005	24.3184	0.0012	841.8788	0.9857
6	identity	0.100	(10,)	0.0016	24.3284	0.0016	840.8928	0.9957
7	identity	0.100	(50, 10)	0.0011	24.3040	0.0020	841.3557	0.9848
8	identity	0.100	(100, 10, 10)	0.0005	24.3185	0.0012	841.8880	0.9857
9	logistic	0.001	(10,)	0.0065	24.2891	0.0065	836.7839	0.9939
10	logistic	0.001	(50, 10)	0.0372	23.8589	0.0373	810.9764	0.9744
11	logistic	0.001	(100, 10, 10)	0.0323	23.8875	0.0323	815.0769	0.9709
12	logistic	0.010	(10,)	0.0065	24.2905	0.0065	836.7831	0.9939
13	logistic	0.010	(50, 10)	0.0078	24.2703	0.0078	835.7256	0.9912
14	logistic	0.010	(100, 10, 10)	0.0066	24.2931	0.0066	836.7160	0.9932
15	logistic	0.100	(10,)	0.0063	24.2943	0.0064	836.9252	0.9941
16	logistic	0.100	(50, 10)	0.0054	24.3127	0.0054	837.7295	0.9943
17	logistic	0.100	(100, 10, 10)	0.0057	24.3030	0.0057	837.5042	0.9933
18	tanh	0.001	(10,)	0.0099	24.2383	0.0099	833.9039	0.9910
19	tanh	0.001	(50, 10)	0.0253	23.9728	0.0254	820.9312	0.9781
20	tanh	0.001	(100, 10, 10)	-0.0000	24.3866	0.0000	842.2672	0.9970
21	tanh	0.010	(10,)	0.0098	24.2396	0.0098	834.0017	0.9911
22	tanh	0.010	(50, 10)	0.0250	23.9807	0.0251	821.1937	0.9814
23	tanh	0.010	(100, 10, 10)	0.0000	24.3865	0.0000	842.2649	0.9970
24	tanh	0.100	(10,)	0.0083	24.2549	0.0083	835.2370	0.9916
25	tanh	0.100	(50, 10)	0.0432	23.7397	0.0433	805.9058	0.9700
26	tanh	0.100	(100, 10, 10)	0.0000	24.3867	0.0000	842.2622	0.9971
27	relu	0.001	(10,)	0.0059	24.2641	0.0059	837.3099	0.9906
28	relu	0.001	(50, 10)	0.0083	24.2023	0.0093	835.2972	0.9800
29	relu	0.001	(100, 10, 10)	0.0146	24.1111	0.0154	829.9853	0.9759
30	relu	0.010	(10,)	0.0059	24.2659	0.0059	837.3248	0.9908
31	relu	0.010	(50, 10)	0.0086	24.2001	0.0096	835.0067	0.9802
32	relu	0.010	(100, 10, 10)	0.0149	24.1091	0.0157	829.7373	0.9761
33	relu	0.100	(10,)	0.0028	24.3020	0.0028	839.9253	0.9939
34	relu	0.100	(50, 10)	0.0085	24.2017	0.0095	835.1495	0.9801
35	relu	0.100	(100, 10, 10)	0.0145	24.1182	0.0153	830.0470	0.9767

Table 4.3: Hyperparameter optimization via grid search on first policy iteration in the working example.

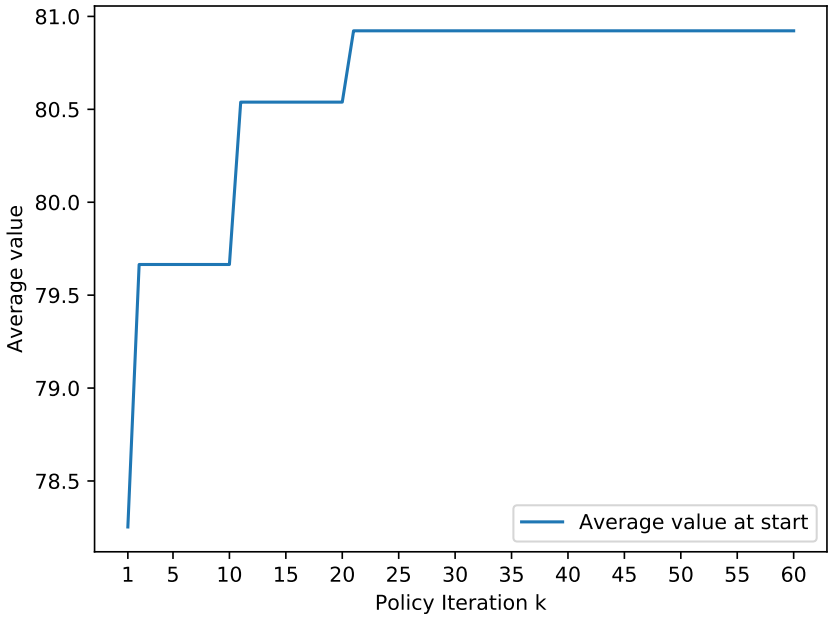


Figure 4.3: Average value at start (y-axis) for the evolution via $K = 60$ policy iterations (x-axis).

That some resources are still left over after the time horizon can be seen in Table 4.4. Again, slight changes can be seen after 10 and after 20 iterations. Almost every time, resource 1 is used completely in the first iteration, this is changed to using less[14] of resource 1 during iterations $2-10$, even fewer during iterations $11-20$ and again slightly more during the final round of iterations. This happens analogously for resources 2 and 4. Resource 3 on the other hand, has less and less capacity remaining until iteration 20. Finally, on average 1.08 of resource 3 remain in the end.

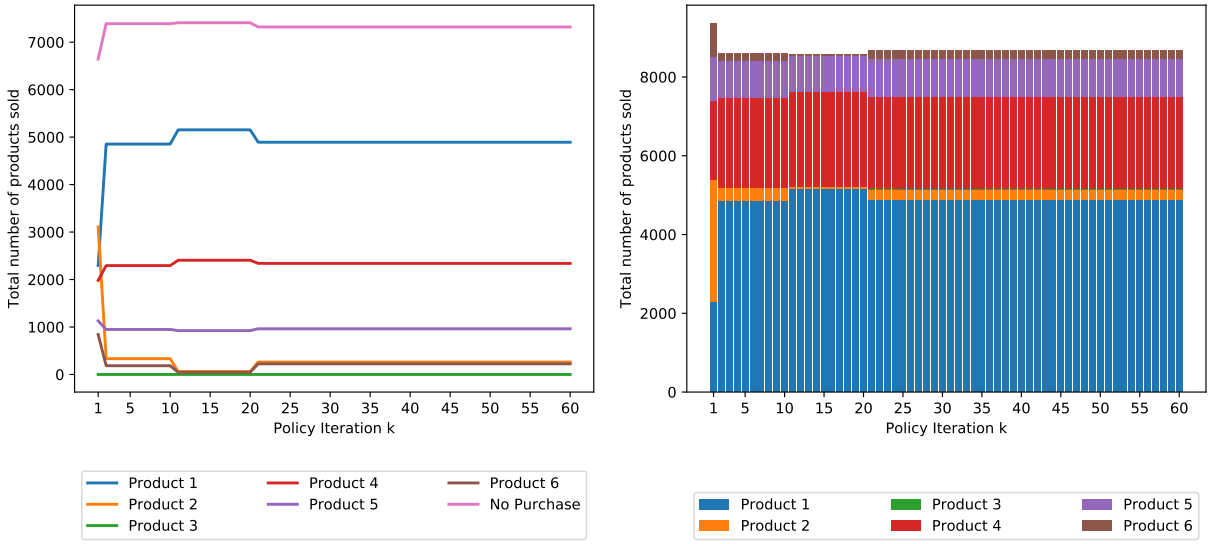
	Resource 1	Resource 2	Resource 3	Resource 4
1	0.19	2.95	1.52	6.59
2-10	1.29	3.77	1.13	6.82
11-20	1.42	3.94	0.99	6.84
21-60	1.28	3.72	1.08	6.80

Table 4.4: Average remaining capacities via $K = 60$ policy iterations in the working example.

These slight changes can also be seen by looking at Figure 4.4, which depicts the total amount of purchased products per iteration. Figure 4.4a includes the "no-purchases"

[14]Note the equivalence of *using less* of one resource leads to *more* left over capacity.

explicitly, while Figure 4.4b depicts only purchased products stacked over each other, leading the total height of each cumulated bar to represent the total amount of purchased products. Note that summing up all purchase actions (including "no-purchase"), leads to a total of $16,000 (= 20 \cdot 800 = T \cdot I)$ in each polity iteration k. Again, the pattern of the value plot (Figure 4.3) and of the remaining capacities (Table 4.4) can be identified. There are slight changes after 10 and after 20 iterations, but mostly no changes between policy iterations. Product 1 is purchased most during iterations $11 - 20$, while the cheap product 2 and the medium priced product 6 are almost never purchased during these iterations (62 and 46 times). Product 3 is never purchased at all (which shouldn't come as a surprise as no customer segment considers this product). The number of no-purchases varies slightly (from $7,389$ over $7,409$ to $7,320$), to ultimately be less then for API ($7,320$ vs. $7,368$).



(a) With no-purchases.

(b) Solely products.

Figure 4.4: Evolution of purchased products (y-axis) via $K = 60$ iterations (x-axis).

A major difference in comparison to API-lc can be seen by taking a look at the offered sets of products as shown in Table 4.5 (and compared to Table 3.3). NN offers a much greater variety than API-lc, but both clearly prefer the set $\{1, 3, 4, 5\}$. It is interesting to see how NN changes its offered sets of products over time. There are sets such as $\{4, 5\}$ that are just offered in the first policy iteration and never again. Other sets such as $\{1, 3, 6\}$ is included as a possibility in the second policy iteration and then offered less and less. Sets such as $\{3\}$ enter just at the eleventh policy iteration and remain. And there, we observe sets such as $\{1, 3, 4, 6\}$ that enter the considered sets in the second iteration, are

	[6]	[5]	[4]	[4,5]	[3]	[3,6]	[3,5]	[3,5,6]	[3,4]	[3,4,6]	[3,4,5]	[3,4,5,6]
1	0	4	0	19	0	0	354	0	0	0	3039	0
2-10	70	128	894	0	0	4	4	41	0	0	28	0
11-20	75	172	790	0	4	14	0	5	0	0	102	5
20-60	75	172	790	0	4	14	0	5	0	0	102	5

	[1]	[1,3,6]	[1,3,5,6]	[1,3,4]	[1,3,4,6]	[1,3,4,5]	[1,3,4,5,6]	[1,2,5]
1	0	0	0	0	0	0	0	2
2-10	1283	23	339	9	10	11107	312	0
11-20	1835	9	45	0	0	12490	141	0
21-60	1143	1	574	3	1	10878	785	0

	[1,2,4,5]	[1,2,3,6]	[1,2,3,5,6]	[1,2,3,4,6]	[1,2,3,4,5]
1	12	0	573	0	0
2-10	0	58	205	0	593
11-20	0	0	14	0	53
21-60	0	7	107	3	536

Table 4.5: Sets offered via $K = 60$ policy iterations in the working example.

not offered during iterations $11 - 20$ and then included again from iteration 21 onwards. This is already an indication of NN being much more flexible than API-lc.

As we have seen that not much is changing during certain phases of iterations, one might suspect that nothing is changing. But we now want to analyse the number of iterations needed to train the neural network in each policy iteration, depicted in Figure 4.5. We clearly see that there is change happening as the number of iterations needed to train the model is changing at random over the course of different policy iterations k.

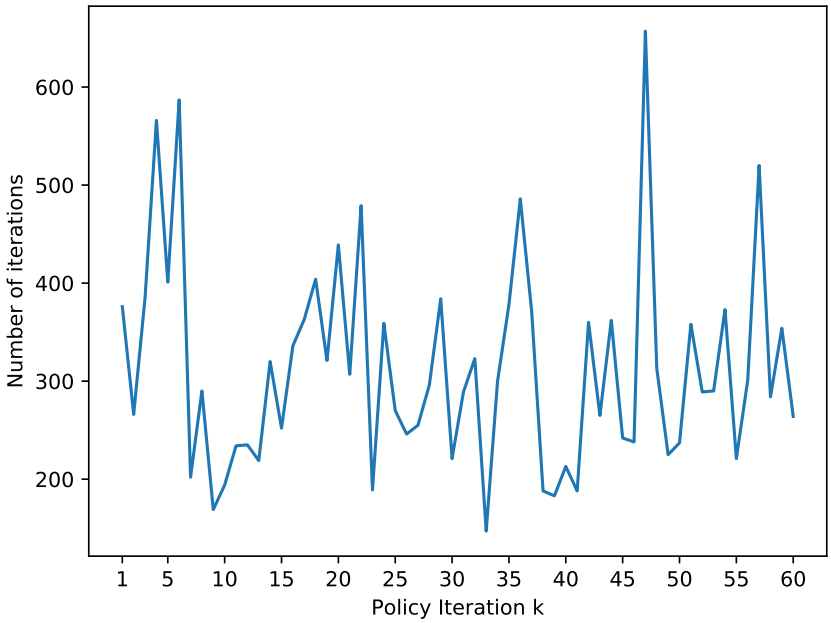


Figure 4.5: Line plot of number of iterations for fitting the neural network over the different policy iterations k (x-axis).

We finally take a look at the offered set as can be seen in Figure 4.6. This figure presents the optimal offersets to be offered at any time (x-axis) and for any combination of resource capacities (y-axis) in a separate colour. By "combination of resource capacities", we refer to the following: As the vector of starting capacities for resources is given by $(8, 4, 4, 8)$ and the least amount of capacity is zero, there are a total of $9 \cdot 5 \cdot 5 \cdot 9 = 2025$ combinations possible. We can clearly see that our neural network is much more flexible than ADP. NN offers more distinct sets of products and also for one given capacity combination, the offerset might change depending on the current time. This is a great difference when compared to the straight horizontal lines occuring for ADP (Figure 3.7). This flexibility is a promising result for further application of NN in the context of revenue management.

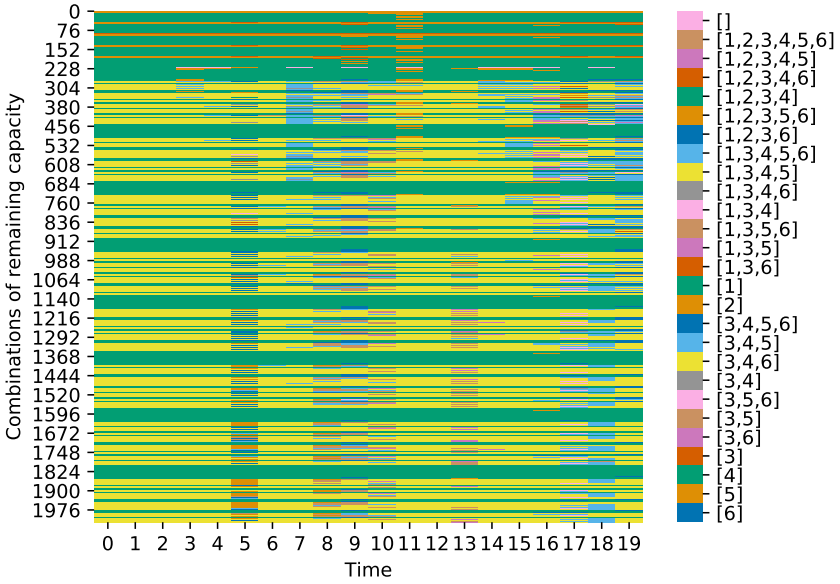


Figure 4.6: Categorical map of offersets for all combinations of remaining capacity for the final network.

Chapter 5

Comparison of different policies

This chapter introduces the concepts applied to compare the different policies. Section 5.1 presents the process of how the evaluation data is generated. Section 5.2 explains some theory on statistical testing which policy performs best and Section 5.3 finally compares the various policies introduced before on the working example.

5.1 Process of generating evaluation data

In the previous chapter, we introduced various algorithms and solved the problem of how to determine which set of products should be offered given a certain time and given certain capacities of resources remaining. Now, we want to evaluate the trained algorithms. Therefore, we prepare unseen data, which is sometimes also called data based on exogenous information. We set up a total of K sample paths[1], each determined by its associated customer stream and sales stream. Of the K *customer streams*, each consists of T time periods. So, all customer streams are represented by a matrix $\in \mathbb{R}^{K \times T}$ of uniformly distributed random variables. In the same way, a matrix of random numbers is used for the *sales stream*. These two entities determine the sample paths, are kept fixed in the whole analysis and used for every algorithm.

Every single algorithm now runs for a given setting (possibly the starting capacities or no-purchase preference change among settings) on each of the K scenarios. It starts at $t = 0$ with capacity $\boldsymbol{c}^0$. The offerset is determined by the trained algorithm and a sales process is simulated given the current value of the customer stream and sales stream random variables. Then, the time value t is incremented and if a product was sold, the

[1] We want to point out that K refers to the total number of sample paths in this chapter and not to the number of policy iterations as in training via approximate policy iteration.

capacity vector is adjusted in accordance with the resources needed. In such a manner, all time periods are evaluated and it is kept track of the offersets offered, products sold, left over capacities and value generated.

We now want to point out some peculiarities of each algorithm.

- ES: The exact solution uses a lookup table to explicitly lookup which offerset is optimal for a given state $(t, \boldsymbol{c})$.
- CDLP: The choice based deterministic linear program determines at the start, i. e. at $(0, \boldsymbol{c}^0)$, for how many time periods which offersets should be offered and allows for non integer (continuous) time periods. As we have discrete time periods, we offer a particular offerset for the assigned time periods, rounded to the next integer value. If rounding does not lead to an admissable assignment of offersets to time periods[2], this is adjusted manually and pointed out in the document. Furthermore, as time evolves, some products might not be allowed to be offered any longer (not all resources are available any longer). If such a thing occurs, the affected product is simply removed from the offerset.
- API-lc: Given a particular state $(t, \boldsymbol{c})$, the offerset is determined with the associated value of $\boldsymbol{\pi}$, derived by API linear concave as outlined in Section 3.2.1.
- API-plc: Given a particular state $(t, \boldsymbol{c})$, first the corresponding value of $\boldsymbol{\pi}$ is calculated, depending on the current level of capacity for each resource h (remember the bins for each resource). The values of $\pi_{t,hs}$ were derived by API piecewise linear concave as outlined in Section 3.2.2.
- API-pl: Given a particular state $(t, \boldsymbol{c})$, first the corresponding value of $\boldsymbol{\pi}$ is calculated, depending on the current level of capacity for each resource h (remember the bins for each resource). The values of $\pi_{t,hs}$ were derived by API piecewise linear as outlined in Section 3.2.3.
- linReg: Given a particular state $(t, \boldsymbol{c})$, the offerset is determined with the associated value of $\boldsymbol{\pi}$, derived by linear regression as outlined in Section 4.1. Parameters were learned closely similiar to API-lc, just without the constraints on the parameters θ and $\boldsymbol{\pi}$.

[2]The sum of rounded assigned time periods is either greater or smaller then the total number of time periods.

- NN-hyper: Given a particular state $(t, \boldsymbol{c})$, the offerset is determined with the usage of a neural network as outlined in Section 4.3. The network structure is taken of the hyperparameter optimization: tanh as activation function, $\alpha = 0.1$, and two hidden layers, one with 50 nodes and the second one with 10 nodes.

- NN-given: Given a particular state $(t, \boldsymbol{c})$, the offerset is determined with the usage of a neural network as outlined in Section 4.3. The network structure is chosen arbitrarily: The logistic function as activation function, $\alpha = 0.1$, and one hidden layer with 10 nodes.

For all examples, we use $K = 5,000$ and T as specified in the example.

5.2 Theory on testing

The main measure determining the goodness of a policy is the value it generates over the time horizon. But to be confident in the statement that one policy is better then another, we need to apply a statistical test and will introduce two statistical tests in the following. Section 5.2.1 presents the paired two sample t-test, while Section 5.2.2 introduces permutation tests.

5.2.1 Paired two sample t-test

Our goal is to evaluate two different policies. One policy is considered better then another policy if it generally leads to higher total revenues. To put this vague phrasing into a scientifically sound comparison, we use an *approximate two sample test*, which we will describe here in more detail. This overview and notation is mainly based on Ch. 14.7 of Bamberg et al. (2011) with extensions as in Ch. 11.3 of Fahrmeir et al. (2007).

As elaborated in Section 5.1, we have a total of K sample paths. A certain policy applied to one particular sample path will generate a value of v, i. e. starting at time $t = 0$ with capacities $\boldsymbol{c}^0$ and the exogenous information associated with this sample path (customer stream and sales stream), a total value of v is generated. Thus, policy A applied to all sample paths generates the vector of values $\boldsymbol{v}^A \in \mathbb{R}^K$. For policy B, $\boldsymbol{v}^B$ is determined accordingly. All components of the vectors are realizations of the underlying random variables $V^A \sim F^A$, resp. $V^B \sim F^B$ that follow a distribution F according to their policies. As both policies are based on the same exogenous information, the samples are

dependent and a *paired samples t-test* can be used. We end up with two observations of size K

$$v_1^A, \ldots, v_K^A \quad \text{respectively} \quad v_1^B, \ldots, v_K^B \ .$$

Let $\bar{\boldsymbol{v}}^A$ and s_A^2 resp. $\bar{\boldsymbol{v}}^B$ and s_B^2 be the empirical mean and empirical variance. Furthermore, μ^A and σ_A^2 denote the expected value and variance of v_i^A, respectively μ^B and σ_B^2 denote the expected value and variance of v_i^B. To test the hypothesis "policy A is better then policy B", we use the following null and alternative hypothesis:

$$H_0 : \mu^A \leq \mu^B \quad \text{versus} \quad H_1 : \mu^A > \mu^B \ .$$

As $\bar{\boldsymbol{v}}^A$ and $\bar{\boldsymbol{v}}^B$ are unbiased estimators of μ^A and μ^B, the difference $D = \bar{\boldsymbol{v}}^A - \bar{\boldsymbol{v}}^B$ can be used to test the hypothesis. A large value of D represents the data to be in favour of H_1.

As the sample size K satisfies $K > 30$ and the distribution of v_i^A or v_i^B is unknown, we follow the forth row of Fig. 51 on p. 193 of Bamberg and Baur (2001) to derive for the test statistic

$$T = \frac{\bar{\boldsymbol{v}}^A - \bar{\boldsymbol{v}}^B}{\sqrt{\frac{s_A^2 + s_B^2}{K}}} \sim N(0;1) \ .$$

With this knowledge, we can compute the p-value according to $p = 1 - \Phi(T)$ with $\Phi(\cdot)$ being the standard normal cumulative distribution function. Note: The p-value can be seen as evidence against H_0. A small p-value represents strong evidence against H_0 and the null hypothesis should be rejected if the p-value is below a significance threshold α.

We use $\alpha = 0.05$ to claim one policy being significantly better then another.

5.2.2 Permutation test

The previously introduced t-test is used to test hypothesis concerning the mean of distributions F_i's and Shao (2008) has shown that "the two-sample t-tests are UMPU [uniformly most powerful unbiased] [...] under the assumption that F_i's are normal with the same variance". As we are unsure about the class of our distributions F^A and F^B, but pretty sure that the variances are not the same, we should also consider another class of tests. Nonparametric tests allow for any distribution and we present an applicable member of this class, the so called *permutation test*, in the following. This subsection is based on the theoretically oriented textbook by Shao (2008) and on Rizzo (2017), which focusses more on computational specialities.

Permutation tests build on resampling and are closely related to bootstrap. But while ordinary bootstrap resamples with replacement, permutation tests resample without replacement. We consider the problem of testing

$$H_0 : F^A = F^B \quad \text{versus} \quad H_1 : F^A \neq F^B \ ,$$

with F^A and F^B being the distributions associated with policies A and B; their distribution class is unknown. Under the null hypothesis, the samples $\boldsymbol{v}^A$ from F^A and $\boldsymbol{v}^B$ from F^B as well as any pooled sample, are all random samples of the same distribution F^A.

The idea of permutation sampling now lies in the following. The actually realized observations $v_1^A, \ldots, v_K^A$ and $v_1^B, \ldots, v_K^B$ can be represented by one (combined) vector $\hat{\boldsymbol{v}}^* = (v_1^A, \ldots, v_K^A, v_1^B, \ldots, v_K^B)$. Under the null hypothesis, this particular vector is just one realization of $2K$ samples chosen from F^A. All other realizations are also equally likely, i. e. have probability

$$\frac{1}{\binom{2K}{K}} = \frac{K!K!}{(2K)!}$$

as also stated by the permutation lemma in Efron and Tibshirani (1998). This allows us to enumerate all possible observations in the set $\{\hat{\boldsymbol{v}}^{(j)} | \text{with } \hat{\boldsymbol{v}}^{(j)} \text{ being one permutation of } \hat{\boldsymbol{v}}^*, j = 1, \ldots, \binom{2K}{K}\}$.

We can now calculate a statistic T on any realization $\hat{\boldsymbol{v}}$ and its cumulative distribution function (cdf) is given by

$$F_T(t) = P(T \leq t) = \binom{2K}{K}^{-1} \sum_{j=1}^{2K} I(T(\hat{\boldsymbol{v}}^{(j)}) \leq t) \ ,$$

with $I(\cdot)$ being the indicator function, i. e. $I(x) = 1$ if x is `true` and $I(x) = 0$ if x is `false`.

This can be used to arrive at the achieved significance level (ASL), i. e. p-value, of the observed statistic $T(\hat{\boldsymbol{v}}^*)$. The ASL is given by

$$P(T \geq T(\hat{\boldsymbol{v}}^*)) = \binom{2K}{K}^{-1} \sum_{j=1}^{2K} I(T(\hat{\boldsymbol{v}}^{(j)}) \geq T(\hat{\boldsymbol{v}}^*)) \ .$$

This establishes the theory needed on permutation tests. But as the total number of permutations is factorial[3], it is infeasible to use all possible permutations. It is sufficient to use a (large) set of random permutations of $\hat{\boldsymbol{v}}^*$.

[3]Note that factorial $n!$ increases even faster than exponential 2^n. In our setting, there are $\binom{10000}{5000} = 1.59 \cdot 10^{3008}$ permutations.

In our case, we use the difference of means as test statistic and $10,000$ random permutations.

5.2.3 Sign test

The previous permutation test allowed us to test whether the distributions belonging to two policies differ. Now, we want to test whether one policy performs indeed better then another one and introduce the *sign test* formally in the following.

Consider the random variables V^A and V^B belonging to policy A and to policy B. To test the hypothesis "policy A is better then policy B", we use the following null and alternative hypothesis:

$$H_0 : P(V^A > V^B) \leq P(V^A < V^B) \quad \text{versus} \quad H_1 : P(V^A > V^B) > P(V^A < V^B) \ .$$

The idea of the rank test is based on the following: If the events $V^A > V^B$ and $V^A < V^B$ are equally likely, than also roughly the same amount of sample realizations $v_i^A > v_i^B$ and $v_i^A < v_i^B$ are to be expected. Thus, a sample realization is opposing our null hypothesis $P(V^A > V^B) \leq P(V^A < V^B)$ if amongst the pairs fulfilling $v_i^A \neq v_i^B$ there are mostly pairs with $v_i^A > v_i^B$.

To arrive at our final test, we let m denote the number of pairs fulfilling $v_i^A = v_i^B$ and let $c \coloneqq T(\boldsymbol{v}^A, \boldsymbol{v}^B) = |\{i : v_i^A > v_i^B\}$, the number of sample paths, for which policy A performed better than policy B. As we assume the events $V^A > V^B$ and $V^A < V^B$ to be equally likely, a binomial $B_{K-m;0.5}$-distribution is used and the random variable C follows this distribution. We arrive at a p-value of

$$p = P(C > c) = 1 - P(C \leq c) = 1 - B_{K-m;0.5}(c) = B_{K-m;0.5}(K - m - c - 1) \ ,$$

where we applied probability of the compliment, used the probability distribution of C and exploited the symmetry of the binomial distribution[4]. Note furthermore that the binomial distribution can be approximated by a normal distribution for large sample sizes $(n - m)$.

[4]Note that $1-B_{n;p}(x) = 1-P(X \leq x) = P(X > x) = P(X \geq x+1) = P(X \leq n-(x+1)) = B_{n;p}(n-x-1)$ for $X \sim B_{n;p}$.

5.3 Comparison applied to working example

In the following, we apply the previously introduced ways of comparing different policies to our working example. We start of with comparing the values generated, continue with offersets offered, products sold and what capacities remained in the final period, and close with statistical tests on which policy is better then another.

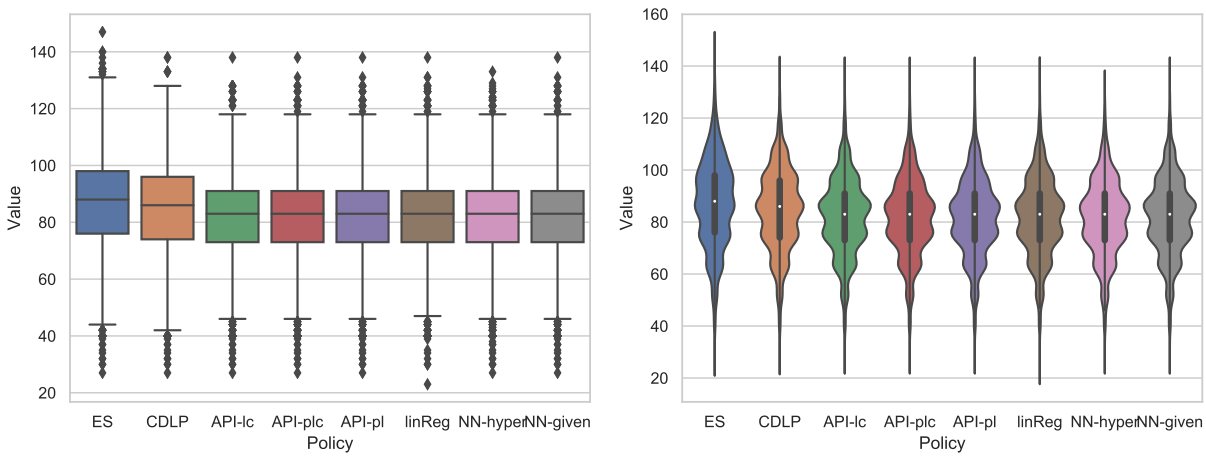
5.3.1 Value generated

A good first overview on the different policies is presented by Figure 5.1. Figure 5.1a presents a classic boxplot of the values generated by each policy. It can clearly be seen that ES performs very well and achieves the overall maximum. Also CDLP performs well and one cannot tell yet whether it is significantly worse then ES. The other algorithms also generate comparable values to ES and CDLP and perform similarly amongst themselves, but some distinctions are visible. Amongst the API-methods, API-plc and API-pl have more outlier on the upper end than API-lc. linReg has one distinct outlier on the lower end and the range of achieved values is smallest for NN-hyper. The narrower boxes of the modern optimization techniques indicate that those strategies have less variance among its generated values as compared to ES and CDLP. Even more information on the distribution of the generated values by each policy can be found in the violinplot presented in Figure 5.1b. ES has more probability mass on the upper side and all policies show quite some bumps, caused by certain values being generated with higher likelihood.

The previous observations are underlined by the summary statistics presented in Table 5.1. It depicts that indeed $K = 5,000$ sample paths are used for evaluation for all methods. ES generates the highest mean (87.11) and also the highest total revenue (147). Interestingly, the minimum value is the same for all policies (27). CDLP ranks second on highest mean. All other methods have very similar menas of generated values. Interestingly, NN-given performs slightly better than NN-hyper on average and it performs as well as the best API-method, which is the (simple) API-lc. But note again, that the means are pretty close together and one cannot tell with (statistical) confidence if ES is indeed the optimal policy.

5.3.2 Offersets offered

The offered sets of products during an average sample path are depicted in Figure 5.2. For every policy, each sample path is looked at and within the sample path, it is counted how



(a) Boxplot (b) Violinplot

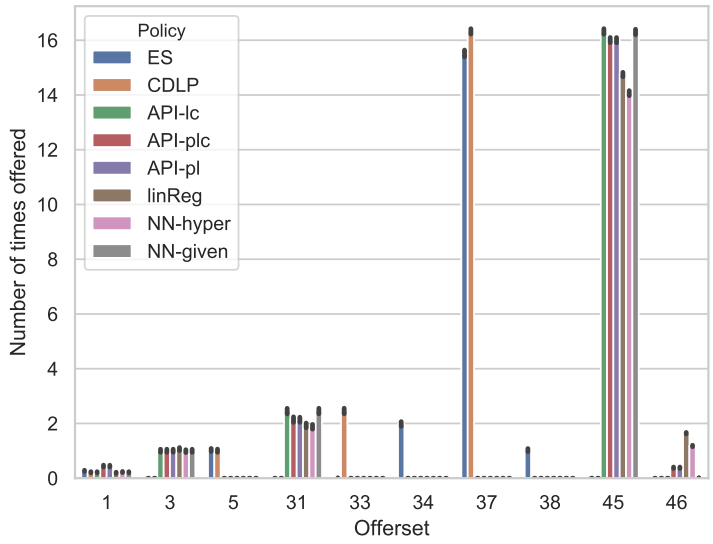
Figure 5.1: Box- and Violinplot to compare different policies on the working example by their generated values.

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
count	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00
mean	87.11	84.45	81.86	81.68	81.69	81.70	81.39	81.86
std	16.86	15.39	14.65	14.56	14.57	14.81	14.57	14.65
min	27.00	27.00	27.00	27.00	27.00	23.00	27.00	27.00
25%	76.00	74.00	73.00	73.00	73.00	73.00	73.00	73.00
50%	88.00	86.00	83.00	83.00	83.00	83.00	83.00	83.00
75%	98.00	96.00	91.00	91.00	91.00	91.00	91.00	91.00
max	147.00	138.00	138.00	138.00	138.00	138.00	133.00	138.00

Table 5.1: Summary statistics to compare different policies on the working example by their generated values.

often a certain set of products was offered. A set is just considered as relevant if it has been offered in more than 0.5% of all possibilities (the caption explains the details). The mean number of periods each set is offered is visualized by the height of the bar and the 95% confidence interval of the number of periods offered is represented by a black bar on top of the coloured bars. The offersets are indexed and the corresponding tuple together with the corresponding set of products is depicted to the right of the barplot. One realizes that both ES and CDLP are in favour of offering products $\{1, 4, 5\}$, while the other methods mainly offer $\{1, 3, 4, 5\}$. Interestingly, linReg and NN-hyper offer this set much less than the other modern methods. Furthermore, ES is the only method to also offer $\{1, 4, 5\}$ and $\{1, 4, 5, 6\}$, while CDLP distinguishes itself by being the only method offering $\{1, 5\}$. Note that all

policies identify product 2 as not worthy enough to be offered. In a business setting, the decision maker should look at this product in depth and change for example its price or its reputation to make it more attractive to the customers and thus more profitable for the company.



Index	Offerset Tuple	Offerset Items
1	(0, 0, 0, 0, 1, 0)	{5}
3	(0, 0, 0, 1, 0, 0)	{4}
5	(0, 0, 0, 1, 1, 0)	{4,5}
31	(1, 0, 0, 0, 0, 0)	{1}
33	(1, 0, 0, 0, 1, 0)	{1,5}
34	(1, 0, 0, 0, 1, 1)	{1,5,6}
37	(1, 0, 0, 1, 1, 0)	{1,4,5}
38	(1, 0, 0, 1, 1, 1)	{1,4,5,6}
45	(1, 0, 1, 1, 1, 0)	{1,3,4,5}
46	(1, 0, 1, 1, 1, 1)	{1,3,4,5,6}

Figure 5.2: Barplot to compare different policies on the working example by their offered products. Offersets are just depicted if they are offered more than 0.5% of all possibilities (i. e. sets that are offered more than $4,000$ times considering all $800,000 = 8 \cdot 5000 \cdot 20$ possibilities to be offered; 8 policies, 5000 sample paths, 20 time periods).

We also want to analyse the corrections that were needed for CDLP (compare the explanation on 62). Table 5.2 presents summary statistics on the first occurence of the corrections. The first column is on the first ocurrence of 0 corrections. As there is enough capacity to offer any product, no correction is needed for any sample path (count of $5,000$), with the first occurrence of 0 corrections certainly during the first time period (standard deviation of 0 and all quantiles have a value of 1). More interesting are the other two columns. One adaptation occurred in $3,384$ sample paths and on average during time period 15.57. The minimum occurrence at the fifth time period leads to the conclusion that the policy ran out of either resource 2 or 3 (both having a capacity of 4 at start). A second adaptation occurred in 456 instances, on average (considering mean) during time period 18.62 and with a median occurrence at time period 19.

	0	1	2
count	5000.0	3384.00	456.00
mean	1.0	15.57	18.62
std	0.0	3.25	1.37
min	1.0	5.00	14.00
25%	1.0	13.00	18.00
50%	1.0	16.00	19.00
75%	1.0	18.00	20.00
max	1.0	20.00	20.00

Table 5.2: Summary statistic to compare different policies on the working example by their sold products. The columns represent the number of corrections and all values (except for count) refer to the first time period when the adaptation took place, e. g. when two adaptations occurred, the second one happened on average during time period 18.62 and at the earliest during time period 14.

5.3.3 Products sold

The amount of products sold during an average sample path are visualized as a barplot in Figure 5.3. Suprisingly, product 2 is indeed sold by one strategy, namely NN-hyper. One also realizes that product 3 was not sold at all, though it was offered quite some times. But this also should not be surprising, as no customer segment has a preference on this product, as stated in Figure 2.1d. Product 1 was purchased most often, slightly most when modern methods were considered. Product 5 was sold significantly more by ES and CDLP than by the other methods (note that the black bars indicating the corresponding 95% confidence intervals do not overlap). Only ES and sometimes linReg and NN-hyper manage to also sell product 6 occasionally. In most time periods, no purchase occurred. There might be three different causes for this:

- No customer arrived.
- A customer arrived but is not interested in any offered product.
- A customer arrived, is interested in some offered products, but decides not to purchase any.

The three causes for a no-purchase are depicted for each policy in Figure 5.4. One can see that the dominating factor is that indeed no customer arrived during this period, which happened in 3.99 time periods on average. Note that this is exactly in line with the arrival

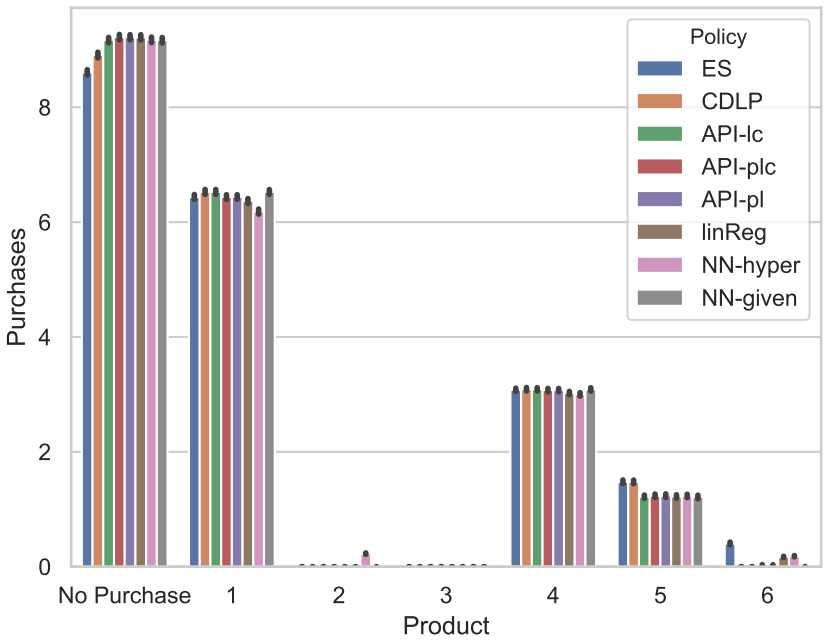


Figure 5.3: Barplot to compare different policies on the working example by their sold products.

probabilities.[5] ES best predicts the interests of arriving customers as it offers a set, which is not of interest to the arriving customer, in just 0.78 time periods on average. API-lc and NN-given are best at predicting the interests of arriving customers amongst the modern methods, but still get it wrong it roughly double the time periods compared to ES (namely 1.5 time periods on average).

5.3.4 Capacities remaining

We also want to take a look at the capacities of resources being left over at the end of the event horizon, which are depicted in Figure 5.5. It can be seen that resource 4 is barely used (initial capacity was 8), which also applies for resource 2 (initial capacity of 4). The different policies use resources quite the same with few exceptions. ES and CDLP use more capacity of resource 4 and ES also achieves to use more of resources 1 and 2.

[5] An arrival probability of 0.2 for no customer per time period leads to an expected no show in $4 = 0.2 \cdot 20$ time periods.

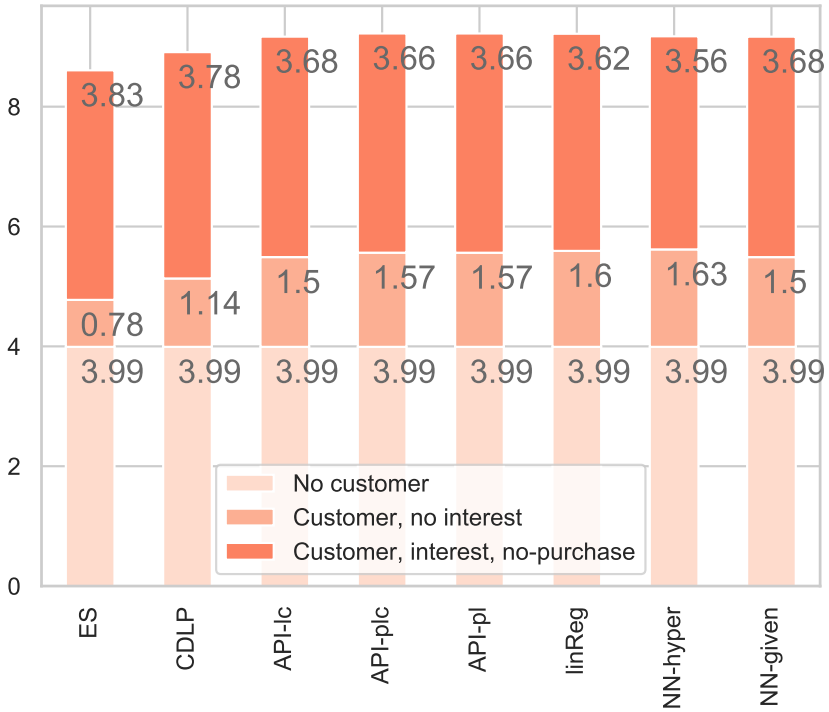


Figure 5.4: Stacked barplot to analyse the causes for no-purchases of different policies on the working example.

5.3.5 Statistical tests

We now compare each policy with each other by their generated values among the sample paths. Table 5.3 presents the p-values of the two sample t-tests when considering the first 100 sample paths (up) or all 5,000 sample paths (down). Clearly, there are no tests performed on the diagonal (comparing one policy with itself makes no sense). All p-values are much smaller then 0.001 leading to the conclusion that the associated means differ significantly. This is already the case for most combinations when considering a sample size of 100, indicating that this sample size as a value for K would have been sufficient. It is directly seen that ES is significantly better than all other methods and also that CDLP is significantly better than all modern methods. But the larger sample size is needed to establish that API-lc is indeed better than most modern methods, but not better than NN-given. Also when looking at the large sample size, one identifies that all strategies are significantly better than NN-hyper. This leads us to the conclusion that our hyperparameter optimization has still room for improvement.

Permutation tests are performed as well and the results presented in Table 5.4, again

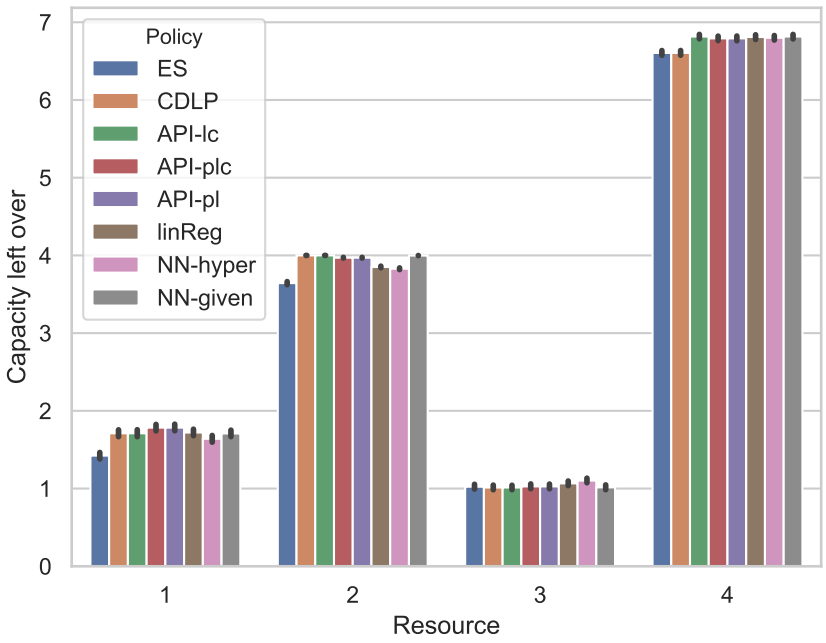


Figure 5.5: Barplot to compare different policies on the working example by their remaining capacities.

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		0.000869453	6.5206e-06	4.45717e-06	4.45717e-06	1.26075e-06	8.41456e-07	5.24802e-06
CDLP	0.000869453		0.000162387	0.00027159	0.00027159	0.000344685	0.000344095	0.000125023
API-lc	6.5206e-06	0.000162387		0.953445	0.953445	0.797587	0.450487	0.319748
API-plc	4.45717e-06	0.00027159	0.953445			0.78144	0.424992	0.776254
API-pl	4.45717e-06	0.00027159	0.953445			0.78144	0.424992	0.776254
linReg	1.26075e-06	0.000344685	0.797587	0.78144	0.78144		0.543275	0.878358
NN-hyper	8.41456e-07	0.000344095	0.450487	0.424992	0.424992	0.543275		0.499607
NN-given	5.24802e-06	0.000125023	0.319748	0.776254	0.776254	0.878358	0.499607	

(a) Sample size 100.

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		1.81419e-209	0	0	0	0	0	0
CDLP	1.81419e-209		2.84886e-216	2.06577e-230	3.60387e-230	3.35872e-186	7.1241e-223	1.47943e-216
API-lc	0	2.84886e-216		1.16936e-05	1.4344e-05	0.00265712	1.54352e-12	0.360471
API-plc	0	2.06577e-230	1.16936e-05		0.317359	0.859206	6.47973e-05	1.75807e-05
API-pl	0	3.60387e-230	1.4344e-05	0.317359		0.883235	5.73007e-05	2.14804e-05
linReg	0	3.35872e-186	0.00265712	0.859206	0.883235		3.84332e-05	0.00320815
NN-hyper	0	7.1241e-223	1.54352e-12	6.47973e-05	5.73007e-05	3.84332e-05		1.97333e-12
NN-given	0	1.47943e-216	0.360471	1.75807e-05	2.14804e-05	0.00320815	1.97333e-12	

(b) Sample size $5,000$.

Table 5.3: p-values for paired two sample t-test with differing sample size. The accuracy is left at its maximum to make difference between the sample sizes visible.

with a sample size of 100 on the upper side and of 5,000 on the lower side. In both settings, 10,000 permutations are randomly drawn. Here, an original sample size of 100 is not enough to prove that any policy is better than another on a 5% significance level, though ES vs. each modern method comes close to it. Here, all policies could be equally good. Only when considering all sample paths, we achieve confidence in the statement that ES is significantly better than CDLP, which itself performs significantly better than the modern methods. A statement on how modern methods compare amongst each others cannot be made by the permutation test. Note that the resulting matrices don't have to be symmetric as different permutations might have been drawn (for each cell).

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		0.428	0.0731	0.0725	0.0775	0.0677	0.0546	0.0798
CDLP	0.4273		0.2918	0.2987	0.3	0.2747	0.2345	0.282
API-lc	0.0722	0.3016		1	1	0.9668	0.8623	0.9885
API-plc	0.0762	0.3045	1		1	0.964	0.8641	0.9831
API-pl	0.0754	0.3027	1	1		0.9669	0.8649	0.9856
linReg	0.0671	0.2778	0.9654	0.9619	0.9617		0.9014	0.9801
NN-hyper	0.0539	0.2275	0.8668	0.8628	0.8632	0.8967		0.8823
NN-given	0.0701	0.2858	0.9889	0.9864	0.9842	0.9827	0.8745	

(a) Sample size 100.

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		0	0	0	0	0	0	0
CDLP	0		0	0	0	0	0	0
API-lc	0	0		0.5407	0.5523	0.5842	0.1094	0.9926
API-plc	0	0	0.5485		0.9962	0.9691	0.3175	0.5513
API-pl	0	0	0.5531	0.9954		0.9761	0.3163	0.5586
linReg	0	0	0.5734	0.9681	0.9741		0.2914	0.5847
NN-hyper	0	0	0.1039	0.3122	0.3195	0.3054		0.1124
NN-given	0	0	0.993	0.5625	0.5581	0.5782	0.1128	

(b) Sample size 5,000.

Table 5.4: p-values for permutation test with differing sample size.

Finally rank tests are applied and the results reported in Table 5.5, also with sample size of 100 on the upper side and 5,000 on the lower side. In both settings, we are confident that ES performed significantly better than any other method and CDLP performed better than all modern methods, as the corresponding p-values are all less than 5%. Interestingly, NN-given and API-lc are identified as the ones next in line. Note that the resulting matrix is not symmetric, as the applied test was directional (one policy being *better* than another).

Overall, the tests showed that ES is indeed the best method, followed by CDLP. The third place is split amongst API-lc and NN-given. This is a good result as it shows that Neural Networks are indeed suitable for capacity control. As NN-given performed better

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		0.000201	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
CDLP	0.998712		0.000000	0.000003	0.000003	0.000035	0.000018	0.000000
API-lc	0.999998	0.999992		0.253906	0.253906	0.037759	0.004780	0.000000
API-plc	0.999998	0.999967	0.500000		0.000000	0.026119	0.002267	0.376953
API-pl	0.999998	0.999967	0.500000	0.000000		0.026119	0.002267	0.376953
linReg	1.000000	0.999844	0.915681	0.938961	0.938961		0.040345	0.876106
NN-hyper	1.000000	0.999936	0.988686	0.994324	0.994324	0.923070		0.984230
NN-given	0.999999	0.999996	0.500000	0.376953	0.376953	0.061039	0.006859	

(a) Sample size 100.

	ES	CDLP	API-lc	API-plc	API-pl	linReg	NN-hyper	NN-given
ES		0.0	0.000000	0.0	0.0	0.0	0.0	0.000000
CDLP	1.0		0.000000	0.0	0.0	0.0	0.0	0.000000
API-lc	1.0	1.0		0.0	0.0	0.0	0.0	0.019287
API-plc	1.0	1.0	1.000000		0.5	0.0	0.0	1.000000
API-pl	1.0	1.0	1.000000	0.0		0.0	0.0	1.000000
linReg	1.0	1.0	1.000000	1.0	1.0		0.0	1.000000
NN-hyper	1.0	1.0	1.000000	1.0	1.0	1.0		1.000000
NN-given	1.0	1.0	0.927002	0.0	0.0	0.0	0.0	

(b) Sample size $5,000$.

Table 5.5: p-values for rank test with differing sample size.

than NN-hyper, our hyperparameter optimization can still be improved.

Chapter 6

Further findings

We also want to state some further findings that occurred while working on the thesis. These do not fit into the regular storyline of answering the question which set of products should be offered at a particular state (time, capacities), but are still worth to mentioned in the overall context. Section 6.1 explains how including the nullset affects CDLP, while Section 6.2 elaborates on how the ordering of the offersets affects ES and thus every determination of optimal offersets.

6.1 CDLP differs depending on the empty set

For validating our implementation of CDLP, we used Example 0 as it was stated in Bront et al. (2009). We introduce the example first and then explore our findings regarding the consideration of the null set.

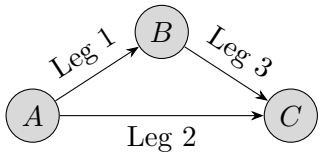
Example 0 represents a small airline network as depicted by Figure 6.1a. Three cities are interconnected by three flights (legs), with capacities stated in Figure 6.1c. The booking horizon consists of $T = 30$ periods and there are five customer segments with preferences as in Figure 6.1d.

Having a total of eight products, there is a total of $2^8 = 256$ possible offer sets[1]. Four constraints in the linear program Equation (2.10), at most four distinct sets are offered over the time horizon.

In order to reproduce the solution as stated in Bront et al. (2009), we run two very similar versions of the CDLP algorithm, just differing in the offersets determining the variables. The optimal result given all possible offersets (i. e. including the empty set) is

[1]Note that in contrast to Bront et al. (2009), we include the empty set as a valid offerset because the company shall offer nothing if there is not enough capacity left to offer any product (otherwise, it is forced to deny any purchase request of a customer, leading to dissatisfaction.)

(a) Airline network.



(b) Products.

Product	Origin-destination	Fare
1	$A \to C$	1,200
2	$A \to B \to C$	800
3	$A \to B$	500
4	$B \to C$	500
5	$A \to C$	800
6	$A \to B \to C$	500
7	$A \to B$	300
8	$B \to C$	300

(c) Resources.

Leg	Capacity
1	10
2	5
3	5

(d) Customers.

Seg.	λ_l	Consideration tuple[a]	Preference vector	No purchase preference	Description
1	0.15	(1, 5)	(5, 8)	2	Price sensitive, Nonstop (A→C)
2	0.15	(1, 2)	(10, 6)	5	Price insensitive, (A→C)
3	0.20	(5, 6)	(8, 5)	2	Price sensitive, (A→C)
4	0.25	(3, 7)	(4, 8)	2	Price sensitive, (A→B)
5	0.25	(4, 8)	(6, 8)	2	Price sensitive, (B→C)

[a]Note that in contrast to Bront et al. (2009), we use the mathematically correct terminology as *tuple* does have an inherent order, while *set* does not (and thus makes no mathematical sense to be combined with a preference vector referring to the order of products in the set).

Figure 6.1: Example 0 of Bront et al. (2009) with $T = 30$ time periods.

stated in Figure 6.2 and excluding the empty set directly from the start leads to results presented in Figure 6.3. Note that both optimization problems terminate at the same optimal objective value (line 22), which should be the case as offering the empty set leads to 0 revenue and thus is not optimal. Also Gurobi becomes aware of this specific feature and presolves this column, i. e. the presolved problem consists purely of variables that are considerable. Astonishingly, even though both problems start with the same specification now (after presolving the empty set, compare to lines 9, resp. 10), they end up at distinct tuples that should be offered (lines 26 and 27), leading to product 4 not being offered if the null set was present at start vs. being offered for 11 time periods if the null set was excluded before the start. This leads us to the insight, that Presolving in Gurobi is something different than just erasing superfluous variables.

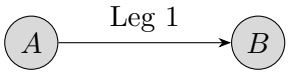
```
Optimize a model with 4 rows, 256 columns and 928 nonzeros
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [6e+01, 5e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [5e+00, 3e+01]
Presolve removed 0 rows and 1 columns
Presolve time: 0.00s
Presolved: 4 rows, 255 columns, 927 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.8638899e+05   5.936988e+02   0.000000e+00      0s
      10    1.1409091e+04   0.000000e+00   0.000000e+00      0s

Solved in 10 iterations and 0.00 seconds
Optimal objective  1.140909091e+04

------------------------------------------------------------


Optimal objective value:
 11409.090909090908

Optimal solution:
Tuple (0, 1, 1, 0, 0, 0, 0, 0)  is offered for a total time of  2.0
Tuple (1, 1, 1, 0, 0, 0, 0, 0)  is offered for a total time of  2.5455
Tuple (1, 1, 1, 0, 0, 1, 0, 0)  is offered for a total time of  25.4545


Product  1 is offered during  28.0000  time periods.
Product  2 is offered during  30.0000  time periods.
Product  3 is offered during  30.0000  time periods.
Product  4 is offered during   0.0000  time periods.
Product  5 is offered during   0.0000  time periods.
Product  6 is offered during  25.4545  time periods.
Product  7 is offered during   0.0000  time periods.
Product  8 is offered during   0.0000  time periods.
```

Figure 6.2: Optimal solution with the null set present at start.

```
Optimize a model with 4 rows, 255 columns and 927 nonzeros
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [6e+01, 5e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [5e+00, 3e+01]
Presolve time: 0.00s
Presolved: 4 rows, 255 columns, 927 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.1657662e+34   5.943074e+32   9.165766e+04      0s
      10    1.1409091e+04   0.000000e+00   0.000000e+00      0s

Solved in 10 iterations and 0.00 seconds
Optimal objective  1.140909091e+04


------------------------------------------------------------


Optimal objective value:
 11409.090909090908

Optimal solution:
Tuple (0, 1, 1, 0, 0, 0, 0, 0)  is offered for a total time of  2.0
Tuple (1, 1, 1, 0, 0, 0, 0, 0)  is offered for a total time of  16.9926
Tuple (1, 1, 1, 1, 0, 1, 0, 0)  is offered for a total time of  11.0074


Product  1 is offered during  28.0000  time periods.
Product  2 is offered during  30.0000  time periods.
Product  3 is offered during  30.0000  time periods.
Product  4 is offered during  11.0074  time periods.
Product  5 is offered during   0.0000  time periods.
Product  6 is offered during  11.0074  time periods.
Product  7 is offered during   0.0000  time periods.
Product  8 is offered during   0.0000  time periods.
```

Figure 6.3: Optimal solution with excluding the null set before start.

(a) Airline network.



(b) Products.

Product	Description	Fare
1	Leg 1 A	1,000
2	Leg 1 B	800
3	Leg 1 C	600
4	Leg 1 D	400

(c) Resources.

Leg	Capacity
1	varying

(d) Customers.

Seg.	λ_l	Consideration tuple[a]	Preference vector	No purchase preference	Description
1	0.5	$(1, 2, 3, 4)$	$(0.4, 0.8, 1.2, 1.6)$	varying	Cheap is good (A→B)

[a]Note that in contrast to Koch (2017), we use the mathematically correct terminology as *tuple* does have an inherent order, while *set* does not (and thus makes no mathematical sense to be combined with a preference vector referring to the order of products in the set).

Figure 6.4: Example single-leg flight as stated in Koch (2017) with $T = 400$ time periods.

6.2 ES offers nothing at start

In the single leg flight scenario of Koch (2017) with no purchase preference of 1, we found that it makes no difference (due to rounding error), whether just the most expensive product is offered or no product at all. Thus, we want to quickly note this result down here and to point out that it led us to put the empty set to the very end of the list of offersets.

First of all, the single leg flight scenario is depicted in Figure 6.4.

We want to estimate the actions in time $t = 1$. After having the final results computed for all capacities in $t = 2$, we depict for capacity $c \in [20]$ the expected value, the optimal offerset and the number of optimal offersets in Table 6.1. Interestingly, the offerset with index 0 is optimal until capacity reaches a value of 13. This is interesting as this offerset corresponds to the set of offering no product at all. Why should this be optimal in any case if we have capacity available? A first clue is that this offerset is not the only optimal offerset, but there are two of them. We will explore this further.

Table 6.2 presents the offersets with the associated purchase probability for each product j given by $\mathrm{pr}(j)$. One can see that both offersets, the one with index 0 and the one with index 8 lead to the optimal expected value of 1000, i. e. offering nothing (offerset with index 0) and offering just the most expensive product (offerset with index 8) both are optimal. This is caused by rounding errors and as the python function `numpy.amax` returns

c	value	optimal offer set	number of optimal offersets
0	0.0	0	0
1	1000.0	0	2
2	2000.0	0	2
3	3000.0	0	2
4	4000.0	0	2
5	5000.0	0	2
6	6000.0	0	2
7	7000.0	0	2
8	8000.0	0	2
9	9000.0	0	2
10	10000.0	0	2
11	11000.0	0	2
12	12000.0	0	2
13	13000.0	8	1
14	14000.0	8	1
15	15000.0	8	1
16	16000.0	8	1
17	17000.0	8	1
18	18000.0	8	1
19	19000.0	8	1
20	20000.0	8	1

Table 6.1: Optimal value together with the optimal offerset in the single leg flight example function for $t = 2$ with capacity as indicated in the first column .

the index of the first maximum (if there are more than one), we faced problems in first implementations.

As it almost never makes sense to offer no product, we switched the offer set with no products to offer at the very end.

index	1	2	3	4	pr(1)	pr(2)	pr(3)	pr(4)	no purchase	exp. value
0	0	0	0	0	0.00	0.00	0.00	0.00	1.00	1000
1	0	0	0	1	0.00	0.00	0.00	0.31	0.69	815
2	0	0	1	0	0.00	0.00	0.27	0.00	0.73	890
3	0	0	1	1	0.00	0.00	0.16	0.21	0.63	810
4	0	1	0	0	0.00	0.22	0.00	0.00	0.78	955
5	0	1	0	1	0.00	0.12	0.00	0.24	0.65	835
6	0	1	1	0	0.00	0.13	0.20	0.00	0.67	893
7	0	1	1	1	0.00	0.09	0.13	0.17	0.61	826
8	1	0	0	0	0.14	0.00	0.00	0.00	0.86	1000
9	1	0	0	1	0.07	0.00	0.00	0.27	0.67	840
10	1	0	1	0	0.08	0.00	0.23	0.00	0.69	907
11	1	0	1	1	0.05	0.00	0.14	0.19	0.62	828
12	1	1	0	0	0.09	0.18	0.00	0.00	0.73	963
13	1	1	0	1	0.05	0.11	0.00	0.21	0.63	852
14	1	1	1	0	0.06	0.12	0.18	0.00	0.65	905
15	1	1	1	1	0.04	0.08	0.12	0.16	0.60	840

Table 6.2: Offersets in the single leg flight example. One row comprises an offerset. The first column indices the offerset. The follwing four columns show which product belongs to one offerset with a 1 in a column indicating that the respective product is offered. The following columns present the purchase probabilities and the last column the expected value when the offerset is indeed offered.

Chapter 7

Conclusion and Outlook

This thesis aimed at solving the capacity control problem under choice behaviour of revenue management via different methods, including modern ones incorporating artificial intelligence. We implemented several methods in *Python* and compared their performance statistically. In the following, we want to summarize the different chapters and give an outlook on further research.

In Chapter 2, we first looked at the problem from three different angles by describing the general setting, specifying it in a concrete airline setting and formulating the task as a dynamic program. Then, we introduced classic solution techniques starting with the exact solution (ES). This method faces computational issues even for small problems and thus, we introduced the upper bound heuristic of choice based linear programming (CDLP). We further elaborated on ways of speeding this method up even more, applied duality theory, and arrived at an algorithm to solve CDLP by Column Generation.

Chapter 3 then continued with introducing the first modern method, namely Approximate Dynamic Programming (ADP). We presented the general algorithm and elaborated on its components Then we formally introduced three direct methods derived of this, namely API linear concave, API piecewise linear concave and API piecewise linear. ADP-lc was exemplarly applied to our example.

Chapter 4 presented our newly derived method, which uses Neural Networks in an Approximate Policy Iteration setting. It started with introducing simple linear regression and then elaborated on general theory on neural networks. Afterwards, we outlined in detail how a neural network can be used in revenue management, before presenting our hyperparameter tuning. Finally, this algorithm was applied to our example.

Chapter 5 presented the theory on the comparison of different methods. It first described the process of generating our evaluation data, before it presented theory on the three sta-

tistical tests: Paired two sample t-test, permutation test and sign test. Afterwards eight algorithms were compared and analysed. This included taking a look at the generated value, at which offersets were offered and which products sold. Also the remaining capacities were looked at and finally the three statistical tests performed. It was found that ES outperformed every other method and CDLP outperformed every modern method. ADP-lc and NN-given outperformed the other modern methods.

Chapter 6 presented two more interesting findings that occured during our research. First, the optimal solution of CDLP is affected by whether or not the empty set is included as an optimization variable when implementing the algorithm in Gurobi. This led us to the conclusion that presolving in Gurobi is more than just eliminating an unnecessary variable. Furthermore the cause for ES offering nothing at start in the single leg example was presented. This happened due to rounding errors by the Python package `numpy` and could be solved by putting the empty offerset at the very end of the list of offersets.

As we have shown that neural networks are a promising path to follow, there are multiple possibilities. We already pointed out that hyperparameter-tuning could improved, which can be done by using more advanced techniques, e. g. Bayesian optimization as presented in the detailed instruction with published code by Will Koehrsen (2013). One could also change the structure of the neural network to use different layers, as e. g. Recurrent Layers or Embedding Layers. For this, one is referred to the Python package `Keras`[1]. Or one could even leave our approach of applying Multi Layer Perceptrons in the setting of Approximate Policy Iteration to use other approaches like temporal difference learning or Q-learning. For this, we refer to Bertsekas (2005). Further promising Python packages include `Stable Baselines`[2], a package of improved implementations of reinforcement learning algorithms and `tf-agents`[3], a library for reinforcement learning in TensorFlow.

We hope that this path is indeed followed by someone and consider this master thesis together with its well written code a good starting point for future research.

[1] https://keras.io/
[2] https://pypi.org/project/stable-baselines/
[3] https://pypi.org/project/tf-agents/

Bibliography

Bamberg, G. and Baur, F. (2001). *Statistik*. Oldenbourgs Lehr- und Handbücher der Wirtschafts- und Sozialwissenschaften. Oldenbourg, München, 11., überarb. aufl. edition.

Bamberg, G., Baur, F., and Krapp, M. (2011). *Statistik*. Lehr- und Handbücher der Wirtschafts- und Sozialwissenschaften. Oldenbourg, München, 16., überarb. aufl. edition.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures.

Bertsekas, D. P. (2005). *Dynamic programming and optimal control*, volume 3 of *Athena scientific optimization and computation series*. Athena Scientific, Belmont, Mass., 3. ed. edition.

Bishop, C. M. (2009). *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, NY, corrected at 8th printing 2009 edition.

Bitran, G., Caldentey, R., and Mondschein, S. (1998). Coordinating clearance markdown sales of seasonal products in retail chains. *Operations Research*, 46(5):609–624.

Borrero, J. S., Gillen, C., and Prokopyev, O. A. (2017). Fractional 0–1 programming: applications and algorithms. *Journal of Global Optimization*, 69(1):255–282.

Bront, J. J. M., Méndez-Díaz, I., and Vulcano, G. (2009). A column generation algorithm for choice-based network revenue management. *Operations Research*, 57(3):769–784.

Courant, R., John, F., and Blank, A. A. (2000). *Introduction to calculus and analysis*. Classics in mathematics. Springer, Berlin, reprint of the 1989 ed. edition.

Diederik P. Kingma and Jimmy Ba (2014). Adam: A method for stochastic optimization.

Domschke, W., Drexl, A., Klein, R., and Scholl, A. (2015). *Einführung in Operations Research*. Springer Gabler, Berlin and Heidelberg, 9., überarbeitete und verbesserte auflage 2015 edition.

Efron, B. and Tibshirani, R. (1998). *An introduction to the bootstrap*, volume 57 of *Monographs on statistics and applied probability*. Chapman & Hall, Boca Raton, Fla., [nachdr.] edition.

Fahrmeir, L., Künstler, R., Pigeot, I., and Tutz, G. (2007). *Statistik: Der Weg zur Datenanalyse*. Springer-Lehrbuch. Springer, Berlin, 6., überarb. aufl. edition.

Feng, Y. and Gallego, G. (2000). Perishable asset revenue management with markovian time dependent demand intensities. *Management Science*, 46(7):941–956.

Gallego, G., Iyengar, G., Phillips, R., and A Dubey (2004). Managing flexible products on a network.

Gallego, G. and van Ryzin, G. (1997). A multiproduct dynamic pricing problem and its applications to network yield management. *Operations Research*, 45(1):24–41.

Galton, F. (1886). Regression towards mediocrity in hereditary stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15:246.

Garey, M. R. and Johnson, D. S. (ca. 2009). *Computers and intractability: A guide to the theory of NP-completeness*. A series of books in the mathematical sciences. Freeman, New York u.a, 27. print edition.

Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, first edition edition.

Gritzmann, P. (2013). *Grundlagen der Mathematischen Optimierung: Diskrete Strukturen, Komplexitätstheorie, Konvexitätstheorie, Lineare Optimierung, Simplex-Algorithmus, Dualität*. Aufbaukurs Mathematik. Springer, Wiesbaden.

Hansen, P., Poggi de Aragão, M. V., and Ribeiro, C. C. (1991). Hyperbolic 0–1 programming and query optimization in information retrieval. *Mathematical Programming*, 52(1-3):255–263.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams (2012). Practical bayesian optimization of machine learning algorithms.

Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122.

Karp, R. M. (2010). Reducibility among combinatorial problems. In Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A., editors, *50 Years of Integer Programming 1958-2008*, pages 219–241. Springer Berlin Heidelberg, Berlin, Heidelberg.

Klein, R. (WS 2017/18). Lecture decision optmization - modellgestützte planung.

Klein, R. and Steinhardt, C. (2008). *Revenue Management: Grundlagen und mathematische Methoden*. Springer-Lehrbuch. Springer, Berlin and Heidelberg.

Koch, S. (2017). Least squares approximate policy iteration for learning bid prices in choice-based revenue management. *Computers & Operations Research*, 77:240–253.

Liu, Q. and van Ryzin, G. (2008). On the choice-based linear programming model for network revenue management. *Manufacturing & Service Operations Management*, 10(2):288–310.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In Unknown, editor, *proceeding of the 6th conference on Natural language learning - COLING-02*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, Mass.

Phillips, R. L. (2011). *Pricing and revenue optimization*. Stanford Business Books, Stanford, Calif., reprinted with corr edition.

Powell, W. B. (2011). *Approximate dynamic programming: Solving the curses of dimensionality*. Wiley series in probability and statistics. J. Wiley & Sons, Hoboken, N.J, 2nd ed. edition.

Prokopyev, O., Meneses, C., Oliveira, C., and Pardalos, P. (2005a). On multiple-ratio hyperbolic 0-1 programming problems. *Pacific Journal of Optimization*, 1.

Prokopyev, O. A., Huang, H.-X., and Pardalos, P. M. (2005b). On complexity of unconstrained hyperbolic 0–1 programming problems. *Operations Research Letters*, 33(3):312–318.

Raschka, S. and Mirjalili, V. (2018). *Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics*. mitp, Frechen, 2., aktualisierte und erweiterte auflage edition.

Rizzo, M. L. (2017). *Statistical computing with R*. Computer science and data analysis series. Chapman & Hall/CRC, Boca Raton, 2. ed. edition.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms.

Shao, J. (2008). *Mathematical Statistics*. Springer Texts in Statistics. Springer, Dordrecht, 2nd ed. edition.

Strauss, A. K., Klein, R., and Steinhardt, C. (2018). A review of choice-based revenue management: Theory and methods. *European Journal of Operational Research*, 271(2):375–387.

Talluri, K. T. and Ryzin, G. J. (2005). *The Theory and Practice of Revenue Management*, volume 68 of *International Series in Operations Research & Management Science*. Springer Science + Business Media Inc, Boston, MA.

Train, K. (2009). *Discrete choice methods with simulation*. Cambridge University Press, Cambridge, second edition edition.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits.

Will Koehrsen (2013). Automated machine learning hyperparameter tuning in python.

Appendix A

Mathematical theory and proofs

Here, we want to bundle thoughts that are more mathematical and would disturb the flow of the master thesis. Appendix A.1 presents some thoughts on exponential smoothing, while Appendix A.2 introduces theory on the Poisson distribution and concludes that the assumption "at most one customer arrives in one time period" is not a restriction at all.

A.1 Exponential smoothing

Lemma 11. (Exponential smoothing) If the optimal solution of the policy iteration optimization model as presented in Equations (3.5) to (3.10) is unique for each policy iteration k, then application of exponential smoothing in every policy iteration results in the same final return variable as using the optimal updates in each iteration and finally returning the overall average.

Exponential smoothing was presented in Equations (3.13) and (3.14) and is shown again without the t index for convenience.

$$\theta^{k+1} = \left(1 - \frac{1}{k}\right)\theta^k + \frac{1}{k}\theta^{update,k} \tag{A.1}$$

$$\boldsymbol{\pi}^{k+1} = \left(1 - \frac{1}{k}\right)\boldsymbol{\pi}^k + \frac{1}{k}\boldsymbol{\pi}^{update,k} \tag{A.2}$$

The other procedure of using the updates directly was presented in Equations (3.11) to (3.12) and is also shown again.

$$\theta^{k+1} = \theta^{update,k} \tag{A.3}$$

$$\boldsymbol{\pi}^{k+1} = \boldsymbol{\pi}^{update,k} \tag{A.4}$$

Proof. Without loss of generality (w. l. o. g.), it suffices to show the statement for θ. We have to show that the iterative exponential smoothing results in the same value as taking the average in the very end. We proof this by induction over the total number of policy iterations K.

Induction hypothesis: For every $K = n\, n \in \mathbb{N}$ it holds that:

$$\theta^{n+1} = \left(1 - \frac{1}{n}\right)\theta^n + \frac{1}{n}\theta^{update,n} = \frac{1}{n}\sum_{i=1}^{n}\theta^{update,i} \ .$$

Induction basis: The statement trivially holds true for $K = 1$.

Induction step: Let the statement hold for $K = n$, i. e.

$$\theta^{n+1} = \left(1 - \frac{1}{n}\right)\theta^n + \frac{1}{n}\theta^{update,n} = \frac{1}{n}\sum_{i=1}^{n}\theta^{update,i} \ .$$

Now, we apply one more step of exponential smoothing, use the induction hypothesis and summation rules to derive our hypothesis and proof Lemma 11.

$$\begin{aligned}
\theta^{n+2} &= \left(1 - \frac{1}{n+1}\right)\theta^{n+1} + \frac{1}{n+1}\theta^{update,n+1} \\
&= \left(1 - \frac{1}{n+1}\right)\left(\frac{1}{n}\sum_{i=1}^{n}\theta^{update,i}\right) + \frac{1}{n+1}\theta^{update,n+1} \\
&= \frac{n}{n+1}\frac{1}{n}\sum_{i=1}^{n}\theta^{update,i} + \frac{1}{n+1}\theta^{update,n+1} \\
&= \frac{1}{n+1}\sum_{i=1}^{n}\theta^{update,i} + \frac{1}{n+1}\theta^{update,n+1} \\
&= \frac{1}{n+1}\sum_{i=1}^{n+1}\theta^{update,i}
\end{aligned}$$

□

Remark 12. Note that the optimization problem to derive the updated values of θ and π depends on their previous values as these are used during the sample runs to determine the offersets and ultimately affect the sample value function $\hat{V}_t^i$. So theoretically there is no (obvious) reason why Lemma 11 shall be applicable. But in my experiments it made no difference, whether exponential smoothing was applied directly or the average taken in the end. The results in this thesis were generated by using exponential smoothing.

A.2 Poisson distribution

Here, we want to analyse our general model assumption of "at most one customer arrives in one time period" and how it is connected to orientating the arrival probability of a customer segment on the parameter of a Poisson distribution. We first present the definition of the Poisson distribution, then establish a lemma on the sum of two Poisson distributions and follow some remarks to conclude that this assumption is not a restriction at all as we are flexible in choosing the model parameters of total time periods T and arrival probability λ at will.

Definition 13. (Poisson distribution) Let the random variable X follow a Poisson distribution with parameter λ: $X \sim \text{Pois}(\lambda)$. Thus, X takes values in $\mathbb{N}_0$ and its probability density function is given by

$$P(X = n) = \frac{\lambda^n \, \mathrm{e}^{-\lambda}}{n!}$$

with e being the Euler number.

Lemma 14. (Sum of two Poisson distributions) Let X_i for $i \in \{1, 2\}$ be two independent random variables that are Poisson distributed with parameter λ_i. This is denoted by $X_i \sim \text{Pois}(\lambda_i)$. Then, the sum is also Poisson distributed with parameter $\lambda_1 + \lambda_2$, which is referred to as the family of Poisson distributions is *closed under convolution*.

Proof. To proof this mathematically, we use that the distribution of a random variable X taking values in the non-negative integers $\mathbb{N}_0$ is also fully described by its *probability generating function* (pgf). The pgf of a discrete random variable X taking values in $\mathbb{N}_0$ is given by

$$G_X(s) = \mathbb{E}(s^X) = \sum_{n \in \mathbb{N}_0} P(X = n) s^n \ , s \in \mathbb{R}$$

with $\mathbb{E}(\cdot)$ denoting the expectation of a random variable.

Consider $X \sim \text{Pois}(\lambda)$. Thus, we calculate

$$\begin{aligned} G_X(s) &= \mathbb{E}(s^X) \\ &= \sum_{n=0}^{\infty} s^n \frac{\lambda^n \, \mathrm{e}^{-\lambda}}{n!} \\ &= \mathrm{e}^{-\lambda} \sum_{n=0}^{\infty} \frac{(s\lambda)^n}{n!} \\ &= \mathrm{e}^{-\lambda} \, \mathrm{e}^{-s\lambda} \\ &= \mathrm{e}^{-\lambda(1-s)} \end{aligned}$$

where we have used the power series for the Euler number: $\mathrm{e}^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ with infinite radius of convergence.

We now apply this to our two random variables $X_i \sim \text{Pois}(\lambda_i)$. Furthermore, we use the fact that the pgf of the sum of independent random variables is the sum of the pgf of the random variables to calculate

$$\begin{aligned} G_{X_1+X_2}(s) &= G_{X_1}(s) + G_{X_2}(s) \\ &= \mathrm{e}^{-\lambda_1(1-s)} \, \mathrm{e}^{-\lambda_2(1-s)} \\ &= \mathrm{e}^{-(\lambda_1+\lambda_2)(1-s)} \end{aligned}$$

Thus, we have shown that for the sum, it holds $X_1 + X_2 = \text{Pois}(\lambda_1 + \lambda_2)$. □

Remark 15. (Expected value of Poisson distribution)
Consider $X \sim \text{Pois}(\lambda)$. We quickly calculate its expected value by

$$\begin{aligned}
\mathbb{E}(X) &= \sum_{n=0}^{\infty} n \cdot P(X = n) \\
&= \sum_{n=0}^{\infty} n \cdot \frac{\lambda^n \, \mathrm{e}^{-\lambda}}{n!} \\
&= \sum_{n=1}^{\infty} n \cdot \frac{\lambda^n \, \mathrm{e}^{-\lambda}}{n!} \\
&= \lambda \, \mathrm{e}^{-\lambda} \sum_{n=1}^{\infty} \cdot \frac{\lambda^{n-1}}{(n-1)!} \\
&= \lambda \, \mathrm{e}^{-\lambda} \sum_{n=0}^{\infty} \cdot \frac{\lambda^{n}}{(n)!} \\
&= \lambda \, \mathrm{e}^{-\lambda} \, \mathrm{e}^{\lambda} \\
&= \lambda
\end{aligned}$$

Remark 16. We now consider T independent random variables $X_t \sim \text{Pois}(\lambda)$ for $t \in \{1, \ldots, T\}$. Knowing that the expected value of a sum of independent random variables equals the expected value of their sum, we calculate the expected value of the convolution $\sum_{t=1}^{T} X_t$:

$$\mathbb{E}\left(\sum_{t=1}^{T} X_t\right) = \sum_{t=1}^{T} \mathbb{E}(X_t) = \sum_{t=1}^{T} \lambda = T\lambda \ .$$

If we half the parameter λ together with doubling T, we get $Y_t \sim \text{Pois}(\lambda)$ for $t \in \{1, \ldots, T\}$ and calculate

$$\mathbb{E}\left(\sum_{t=1}^{2T} Y_t\right) = \sum_{t=1}^{2T} \mathbb{E}(Y_t) = \sum_{t=1}^{2T} \frac{\lambda}{2} = 2T\frac{\lambda}{2} = T\lambda \ .$$

Remark 17. We consider $X \sim \text{Pois}(\lambda)$ and the probability of more than one customer arriving: $P(X > 1) = 1 - P(X = 0) - P(X = 1) = 1 - \frac{\lambda^0 \, \mathrm{e}^{-\lambda}}{0!} - \frac{\lambda^1 \, \mathrm{e}^{-\lambda}}{1!} = 1 - \mathrm{e}^{-\lambda} - \lambda \, \mathrm{e}^{-\lambda} = 1 - \mathrm{e}^{-\lambda} \cdot (1 + \lambda)$. Furthermore, we have

$$\lim_{\lambda \to 0} P(X > 1) = 1 - \mathrm{e}^{-0} \cdot (1 + 0) = 1 - 1 = 0 \ ,$$

leading to the fact, that the restriction of "not more than one customer per time period" really is not a restriction at all as we can change the parameters λ and T in our model to

fit the situation we are facing.

Appendix B

Learnings and best practices

Here, I want to present some learnings and best practices for writing a Master Thesis, structured by some topics.

- Coding
  - choose a proper IDE (Pycharm is powerful, full-stack) to profit from established software development features
  - work with multiple consoles to work in parallel on multiple things and not have to wait for job to be finished
  - include a silent mode in methods to not be distracted by noisy output on console
  - include time stamp when executing scripts and also print it on console to derive rules of thumb on how long it might take
  - choose a proper version management system (Git) to document working progress and to have checkpoints of code that is working
  - choose a suitable file exchange system (LRZ Sync and Share works fine, Git can't store files $>$ 100 MB)
  - save results in a manner, so that they are reproducible as it helps a lot in later stages of work
  - use IPython Notebooks in early stages of development for fast results and presentation to plus discussion with supervisor
  - use remote computer for heavy calculations
  - check and ensure correct signatures when calling methods
  - directly take care of NaN's

- Supervision
  - ensure to be on the agenda of the supervisor
  - establish regular meetings to get into workflow and have pressure to produce results
  - take measures with care if supervision continuous to be suboptimal
- Personal
  - outline the working process incrementally (whole master thesis, months, weeks, days)
  - work in sprints to stay motivated and get tasks done quickly
  - establish checkup points when some results are written down
  - be aware of the progress of work and current status of the final document

Appendix C

Thoughts on implementation

For getting to know Python (the standard programming language when it comes to Machine Learning), I can recommend online courses by DataCamp. The ones looking for a broad overview are referred to the Python Course. And those aiming straight for Machine Learning techniques should take a look at the textboks by Géron (2017) and Raschka and Mirjalili (2018). A good approach on getting started with Machine Learning with might be the corresponding Gurobi Webinar on Machine Learning.

The general outline of the code is given in the following. Scripts indicate that they can be run from the commandline via `python script.py`.

1. Data Handling
    a) *A_data_entry.py*: enter data; different dictionaries for different example settings, dictionary for different variables, numpy arrays for parameter values; check dimensions; pickle data and save it on file system
    b) *A_data_read.py*: methods for reading in data of a given example name
    c) *0_settings.csv*: additional parameters that might change more often
2. Helper
    a) *B_helper.py*: methods needed by all methods
3. Exact solution
    a) *D_exact_solution_multi_leg.py*: script to train the exact solution
    b) *D_exact_solution_multi_leg_evaluation.py*: script to generate the evaluation data for the exact solution
4. CDLP
    a) *CDLP_multi_leg.py*: script to train the CDLP

   b) *CDLP_multi_leg_evaluation.py*: script to generate the evaluation data for the CDLP

5. API-lc
   a) *E_API_multi_leg.py*: script to train the API-lc
   b) *E_API_multi_leg_evaluation.py*: script to generate the evaluation data for the API-lc
6. API-plc
   a) *F_API_plc_multi_leg.py*: script to train the API-plc
   b) *F_API_plc_multi_leg_evaluation.py*: script to generate the evaluation data for the API-plc
7. API-pl
   a) *F_API_pl_multi_leg.py*: script to train the API-pl
   b) *F_API_pl_multi_leg_evaluation.py*: script to generate the evaluation data for the API-pl
8. LinReg
   a) *G_linReg_multi_leg.py*: script to train the LinReg
   b) *G_linReg_multi_leg_evaluation.py*: script to generate the evaluation data for the LinReg
9. NN
   a) *G_NN3_multi_leg.py*: script to train the NN
   b) *G_NN3_multi_leg_evaluation.py*: script to generate the evaluation data for the NN

Some more thoughts:

1. I wanted to produce reproducible results, i.e. my code shall be usable on a larger scope. Logic and Settings shall be separated. Completed code shall be script based and after run, create a folder with its running configuration.
2. Be careful, when storing list or dataframes, as a deepcopy might have to be used.

Appendix D

Data scientist

Figure D.1: Certificate "Data Scientist with Python Track" with 100 learning hours.

This career track on DataCamp helped me to improve my Python coding skills and apply analytical techniques in Python. It comprised the courses:

1. Introduction to Python
2. Intermediate Python for Data Science
3. Python Data Science Toolbox (Part 1)
4. Python Data Science Toolbox (Part 2)
5. Importing Data in Python (Part 1)
6. Importing Data in Python (Part 2)
7. Cleaning Data in Python
8. pandas Foundations
9. Manipulating DataFrames with pandas
10. Merging DataFrames with pandas
11. Analyzing Police Activity with pandas
12. Intro to SQL for Data Science
13. Introduction to Relational Databases in SQL
14. Introduction to Data Visualization with Python
15. Interactive Data Visualization with Bokeh
16. Statistical Thinking in Python (Part 1)
17. Statistical Thinking in Python (Part 2)
18. Joining Data in SQL
19. Introduction to Shell for Data Science
20. Conda Essentials
21. Supervised Learning with scikit-learn
22. Machine Learning with the Experts: School Budgets
23. Unsupervised Learning in Python
24. Machine Learning with Tree-Based Models in Python
25. Deep Learning in Python
26. Network Analysis in Python (Part 1)

Appendix E

Schriftliche Versicherung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbständig und ohne fremde Hilfe verfasst wurde. Alle Stellen, die ich wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen übernommen habe, habe ich als solche gekennzeichnet. Es wurden alle Quellen, auch Internetquellen, ordnungsgemäß angegeben.

Augsburg	30.09.2019
Ort	Datum

Stefan Glogger